

Modul Praktikum Algoritma dan Pemrograman (TI0082)

Semester Genap 2022/2023

Program Studi Informatika

**Yuan Lukito, S.Kom., M.Cs.,
Laurentius Kuncoro Probo Saputro, S.T., M.Eng.,
Matahari Bhakti Nendya, S.Kom., M.T.**



**UNIVERSITAS KRISTEN
DUTA WACANA**

Copyright © 2023 Yuan Lukito, Laurentius Kuncoro Probo Saputro, Matahari Bhakti Nendya

Dipublikasikan oleh:
Program Studi Informatika, Fakultas Teknologi Informasi
Universitas Kristen Duta Wacana, Yogyakarta

Email: yuanlukito@ti.ukdw.ac.id, kuncoro@staff.ukdw.ac.id, didanendya@staff.ukdw.ac.id

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Yogyakarta, Februari 2023

Tim Penulis



Yuan Lukito, S.Kom., M.Cs. menyelesaikan S1 dan S2 di program studi Ilmu Komputer, Universitas Gadjah Mada, Yogyakarta. Saat ini bekerja sebagai dosen di program studi Informatika, Universitas Kristen Duta Wacana. Matakuliah yang diampu adalah Algoritma dan Pemrograman, Struktur Data, Pemrograman Berorientasi Obyek, Supervised Learning dan Internet of Things. Topik riset yang diminati adalah adaptive e-Learning, machine learning, pattern recognition dan quantitative finance.



Laurentius Kuncoro Probo Saputro, S.T., M.Eng. menyelesaikan S1 di program studi Teknik Elektro Universitas Sanata Dharma dan S2 di Teknik Elektro dan Teknologi Informasi Universitas Gadjah Mada. Matakuliah yang diampu adalah Algoritma dan Pemrograman, Teknologi Komputer, Jaringan Komputer, Internet of Things dan Unsupervised Learning. Topik riset yang diminati adalah Internet of Things, Big Data.



Matahari Bhakti Nendya, S.Kom., M.T. menyelesaikan S1 di Teknik Informatika Universitas Kristen Duta Wacana dan S2 Game Technology, Jurusan Teknik Elektro, Fakultas Teknik Industri, Institut Teknologi Sepuluh Nopember. Matakuliah yang diampu adalah Teknologi Komputer, Algoritma dan Pemrograman, Sistem Basis Data, dan Desain Game. Topik riset yang diminati adalah Games Design, Computer Graphic dan Kecerdasan Artifisial.

Kata Pengantar

Modul mata kuliah Praktikum Algoritma dan Pemrograman (TI0082) ini telah berhasil diselesaikan atas berkat dan penyertaan yang diberikan oleh Tuhan yang Maha Kuasa. Modul ini disusun berdasarkan kurikulum Program Studi Informatika 2021.

Modul Praktikum Algoritma dan Pemrograman ini disusun dengan menggunakan pendekatan praktis, yaitu lebih banyak menekankan pada kegiatan praktek menyelesaikan masalah-masalah algoritma dan pemrograman yang umum dijumpai. Oleh karena itu sebagian besar dari modul ini berisi penjelasan sintaks bahasa pemrograman, penyusunan alur program dan contoh kasus beserta penyelesaiannya.

Bahasa pemrograman yang digunakan dalam modul ini adalah bahasa Python. Python merupakan salah satu bahasa pemrograman yang mulai populer sejak tahun 2018 sampai sekarang karena mudah untuk dipelajari, memiliki banyak panduan dan tutorial di Internet serta banyak digunakan dalam bidang Data Science dan Machine Learning yang populer saat ini. Pemilihan bahasa pemrograman Python juga mempertimbangkan kegunaannya untuk mahasiswa dalam menghadapi mata kuliah-mata kuliah berikutnya seperti Struktur Data, Pemrograman Berorientasi Obyek, Pemrograman Web, Kecerdasan Buatan, Supervised Learning, Unsupervised Learning dan beberapa mata kuliah lainnya yang serupa.

Modul mata kuliah Praktikum Algoritma dan Pemrograman ini terdiri dari 14 bab utama, yang dapat disesuaikan pembagiannya untuk perkuliahan selama 1 semester (14 pertemuan). Setiap bab terdiri dari tujuan pertemuan (setiap bab), dasar teori, kegiatan perkuliahan dan latihan mandiri.

Kami harapkan modul praktikum ini dapat menjadi panduan dan membantu proses pembelajaran pada mata kuliah Praktikum Algoritma dan Pemrograman di Program Studi Informatika Universitas Kristen Duta Wacana khususnya, dan dapat digunakan juga untuk Universitas lain yang memiliki kurikulum serupa. Kami juga mengharapkan saran, masukan dan kritik mengenai modul praktikum ini, sehingga dapat terus diperbaiki dan terus disesuaikan dengan kebutuhan penyelenggaraan mata kuliah Praktikum Algoritma dan Pemrograman.

Selamat menikmati proses belajar, semoga tujuan anda dapat tercapai dengan baik. Selamat belajar!

Yogyakarta, 1 Februari 2023

Tim Penulis

Daftar Isi

Sampul	i
Copyright	ii
Kata Pengantar	iv
Daftar Isi	x
Daftar Gambar	xiii
Daftar Tabel	xv

I

Pengenalan Bahasa Pemrograman Python

1	Bahasa Pemrograman Python	1
1.1	Tujuan Praktikum	1
1.2	Alat dan Bahan	1
1.3	Materi	2
1.3.1	Kenapa Python?	2
1.3.2	Menginstall Python 3	3
1.3.3	Menjalankan Python Mode Interaktif	4
1.3.4	Editor untuk Python	5
1.3.5	Menjalankan Script Python di Terminal/Console	6
1.3.6	Mencari Bug dan Memperbaikinya (debugging)	7
1.4	Kegiatan Praktikum	11
1.4.1	Menginstall Package Jupyter Notebook	11

1.4.2	Eksplorasi Python mode Interaktif	13
1.5	Latihan Mandiri	16
2	Variable, Expression dan Statements	19
2.1	Tujuan Praktikum	19
2.2	Alat dan Bahan	19
2.3	Materi	19
2.3.1	Values dan type	19
2.3.2	Variabel	20
2.3.3	Nama Variabel dan Keywords	21
2.3.4	Statements	22
2.3.5	Operator dan Operand	22
2.3.6	Expressions	23
2.3.7	Urutan Operasi	23
2.3.8	Operator Modulus dan String	24
2.3.9	Menangani Input dari Pengguna	24
2.4	Komentar	26
2.5	Kegiatan Praktikum	26
2.5.1	Membuat Variabel	26
2.5.2	Memberikan nilai dalam variabel	28
2.5.3	Mencetak nilai dalam variabel	29
2.5.4	Separator, tipe data dan fungsi type	29
2.6	Latihan Mandiri	30

II

Struktur Kontrol dan Modular Programming

3	Struktur Kontrol Percabangan	35
3.1	Tujuan Praktikum	35
3.2	Alat dan Bahan	35
3.3	Materi	35
3.3.1	Boolean Expression dan Logical Operator	35
3.3.2	Bentuk-bentuk Percabangan	37
3.3.3	Penanganan Kesalahan Input Menggunakan Exception Handling	39
3.4	Kegiatan Praktikum	41
3.4.1	Contoh Masalah-masalah Percabangan	41
3.5	Latihan Mandiri	44
4	Modular Programming	47
4.1	Tujuan Praktikum	47
4.2	Alat dan Bahan	47
4.3	Materi	47
4.3.1	Fungsi, Argument dan Parameter	47
4.3.2	Return Value	49
4.3.3	Optional Argument dan Named Argument	51
4.3.4	Anonymous Function (Lambda)	52

4.4	Kegiatan Praktikum	53
4.4.1	Mendefinisikan Fungsi	53
4.4.2	Argument dan Parameter	54
4.4.3	Anonymous/Lambda Function	55
4.5	Latihan Mandiri	55
5	Struktur Kontrol Perulangan	57
5.1	Tujuan Praktikum	57
5.2	Alat dan Bahan	57
5.3	Materi	57
5.3.1	Definisi Perulangan	57
5.3.2	Bentuk Perulangan for	58
5.3.3	Bentuk Perulangan While	59
5.3.4	Penggunaan Break dan Continue	59
5.3.5	Konversi dari Bentuk for Menjadi Bentuk while	60
5.4	Kegiatan Praktikum	61
5.4.1	Deret Bilangan	61
5.4.2	Penggunaan Break	63
5.5	Latihan Mandiri	64
6	Percabangan dan Perulangan Kompleks	67
6.1	Tujuan Praktikum	67
6.2	Alat dan Bahan	67
6.3	Materi	67
6.3.1	Struktur Percabangan Kompleks	67
6.3.2	Struktur Perulangan Kompleks	73
6.4	Kegiatan Praktikum	77
6.5	Latihan Mandiri	79

III

String dan File

7	Pengolahan String	83
7.1	Tujuan Praktikum	83
7.2	Alat dan Bahan	83
7.3	Materi	83
7.3.1	Pengantar String	83
7.3.2	Pengaksesan String dan Manipulasi String	84
7.3.3	Operator dan Metode String	84
7.3.4	Parsing String	87
7.4	Kegiatan Praktikum	87
7.4.1	Pembahasan Soal 1	88
7.4.2	Pembahasan Soal 2	88
7.4.3	Pembahasan Soal 3	88
7.4.4	Pembahasan Soal 4	89
7.5	Latihan Mandiri	89

8	Membaca dan Menulis File	91
8.1	Tujuan Praktikum	91
8.2	Alat dan Bahan	91
8.3	Materi	91
8.3.1	Pengantar File	91
8.3.2	Pengaksesan File	92
8.3.3	Manipulasi File	94
8.3.4	Penyimpanan File	96
8.4	Kegiatan Praktikum	96
8.5	Latihan Mandiri	97

IV

Type Data Collection

9	Type Data List	101
9.1	Tujuan Praktikum	101
9.2	Alat dan Bahan	101
9.3	Materi	101
9.3.1	Sifat-sifat List	101
9.3.2	Mengakses dan Mengubah isi List	103
9.3.3	Fungsi-fungsi Untuk List	103
9.3.4	List Sebagai Parameter Fungsi	103
9.4	Kegiatan Praktikum	103
9.5	Latihan Mandiri	103
10	Type Data Dictionary	105
10.1	Tujuan Praktikum	105
10.2	Alat dan Bahan	105
10.3	Materi	105
10.3.1	<i>Dictionary</i> sebagai set penghitung (<i>counters</i>)	107
10.3.2	<i>Dictionary</i> dan File	109
10.3.3	<i>Looping</i> dan <i>Dictionary</i>	110
10.3.4	<i>Advanced Text Parsing</i>	111
10.4	Kegiatan Praktikum	113
10.5	Latihan Mandiri	116
11	Type Data Tuples	119
11.1	Tujuan Praktikum	119
11.2	Alat dan Bahan	119
11.3	Materi	120
11.3.1	<i>Tuple Immutable</i>	120
11.3.2	Membandingkan <i>Tuple</i>	121
11.3.3	Penugasan <i>Tuple</i>	122
11.3.4	<i>Dictionaries and Tuple</i>	124
11.3.5	Multipenugasan dengan <i>dictionaries</i>	124

11.3.6	Kata yang sering muncul	125
11.3.7	Tuple sebagai kunci <i>dictionaries</i>	126
11.4	Kegiatan Praktikum	127
11.5	Latihan Mandiri	130
12	Tipe Data Set	133
12.1	Tujuan Praktikum	133
12.2	Alat dan Bahan	133
12.3	Materi	133
12.3.1	Pengenalan dan Mendefinisikan Set	133
12.3.2	Pengaksesan Set	134
12.3.3	Operasi-Operasi pada Set	137
12.4	Kegiatan Praktikum	140
12.4.1	Contoh-contoh Kasus Set	140
12.5	Latihan Mandiri	143

V

Rekursif dan Regular Expression

13	Fungsi Rekursif	147
13.1	Tujuan Praktikum	147
13.2	Alat dan Bahan	147
13.3	Materi	147
13.3.1	Pengertian Rekursif	147
13.3.2	Kelebihan dan Kekurangan	148
13.3.3	Bentuk Umum dan Studi Kasus	148
13.4	Kegiatan Praktikum	149
13.4.1	Problem dan Solusi 1	149
13.4.2	Problem dan Solusi 2	150
13.4.3	Problem dan Solusi 3	151
13.4.4	Problem dan Solusi 4	151
13.4.5	Problem dan Solusi 5	154
13.5	Latihan Mandiri	155
14	Regular Expression	157
14.1	Tujuan Praktikum	157
14.2	Alat dan Bahan	157
14.3	Materi	157
14.3.1	Pengantar Regex	157
14.3.2	Meta Character, Escaped Character, Set of Character, dan Fungsi Regex pada Library Python	159
14.4	Kegiatan Praktikum	159
14.4.1	Penggunaan findall	159
14.4.2	Penggunaan search	162
14.4.3	Penggunaan split	163
14.4.4	Penggunaan sub	163

14.5	Latihan Mandiri	163
15	Revision History	165
15.1	Daftar Revisi Modul Praktikum	165
15.1.1	Semester Genap 2021/2022	165
15.1.2	Semester Genap 2019/2020	165
	Referensi	167
	Index	169

Daftar Gambar

1.1	Logo Python (diambil dari https://www.python.org/)	2
1.2	Python versi 3 di Ubuntu Linux. Terinstall Python versi 3.7.5	3
1.3	Distribusi Anaconda Individual Edition.	4
1.4	Menu Anaconda Prompt setelah terinstall di Windows.	4
1.5	Informasi versi Python akan ditampilkan dan siap menerima perintah.	4
1.6	Menghitung Luas Segitiga dengan bantuan Python	5
1.7	Python Extension untuk Visual Studio Code.	6
1.8	Tampilan Visual Studio Code saat menjalankan script Python.	6
1.9	Tampilan PyCharm saat mengedit file Python.	6
1.10	Menjalankan script Python di Terminal Ubuntu.	7
1.11	Menjalankan script Python pada Visual Studio Code.	8
1.12	Kesalahan yang muncul saat script dijalankan.	9
1.13	Visual Studio Code menemukan kesalahan sintaks sebelum script dijalankan.	9
1.14	Script sudah bisa dijalankan sampai selesai dengan baik.	10
1.15	Hasil sudah sesuai dengan yang diharapkan.	11
1.16	Contoh tampilan Jupyter Notebook.	12
1.17	Instalasi Jupyter Notebook menggunakan pip.	12
1.18	Tampilan awal Jupyter Notebook.	13
1.19	Menu untuk membuat notebook baru.	13
1.20	Tampilan awal notebook baru di Jupyter Notebook.	13
1.21	Penyelesaian untuk masalah usia Trump.	14
1.22	Hasil perhitungan compound interest selama tiga tahun.	15
1.23	Hasil perhitungan compound interest menggunakan formula.	15
1.24	Menggunakan Jupyter Notebook untuk menghasilkan grafik.	16
2.1	Contoh statement dan outputnya	22
2.2	Input - Proses - Output.	25
2.3	Contoh input/output tipe data string.	27

2.4	Contoh input/output tipe data bilangan.	28
2.5	Contoh input/output menggunakan variabel.	29
2.6	Tampilan Contoh print Tipe Data String, Integer dan Float.	30
3.1	Kemungkinan hasil dari boolean expression.	36
3.2	Hasil program kategori usia.	40
3.3	Jika input tidak sesuai yang diharapkan, program akan berhenti.	40
3.4	Hasil program kategori usia.	41
3.5	Menentukan demam atau tidak berdasarkan suhu tubuh.	42
3.6	Menentukan demam atau tidak berdasarkan suhu tubuh.	42
3.7	Hasil dari program Positif-Negatif.	43
3.8	Hasil dari program mencari bilangan terbesar.	44
4.1	Hasil dari pemanggilan fungsi tambah().	49
4.2	Fungsi tambah() belum dikenali karena didefinisikan di bawahnya.	50
5.1	Penggunaan while untuk mengambil input dari pengguna sampai sesuai dengan permintaan.	59
5.2	Penggunaan continue untuk melewati iterasi saat counter(i) bernilai 6.	60
5.3	Deret bilangan fibonacci.	61
5.4	Deret bilangan fibonacci yang kurang dari 100 dan kurang dari 1000.	62
5.5	Deret bilangan konvergen yang dimulai dari 12 dan 5.	63
5.6	Penggunaan break untuk menghentikan perulangan while.	64
5.7	Hasil luaran program IPS mahasiswa	65
6.1	Flowchart Percabangan Kompleks Bentuk 1	68
6.2	Flowchart Percabangan Kompleks Bentuk 2	69
6.3	Flowchart Percabangan Kompleks Bentuk 3	70
6.4	Flowchart Percabangan Kompleks Bentuk 4	71
6.5	Flowchart Percabangan Kompleks Bentuk 5	71
6.6	Flowchart Percabangan Kompleks Bentuk 6	72
6.7	Flowchart Program Menggunakan Break	74
6.8	Flowchart Program Menggunakan Break 2	75
6.9	Flowchart Program Menggunakan Continue	76
6.10	Soal Segitiga	79
7.1	Bentuk string di dalam memory	84
8.1	Secondary Memory di Komputer	92
8.2	Contoh Property File	93
8.3	Ilustrasi Handle File	93
8.4	Contoh File Teks dan Barisnya	94
12.1	Fungsi-fungsi untuk menghapus anggota dari Set.	136
12.2	Operasi-operasi pada Set.	138
13.1	Rumus faktorial	148
13.2	Pseudocode rekursif faktorial	149
13.3	Proses perhitungan faktorial rekursif	149
13.4	Proses perhitungan perkalian rekursif	150
13.5	Rumus Fibonacci	151
13.6	Desimal ke Biner	152
13.7	Desimal ke Oktal	153

13.8	Desimal ke Hexa	154
------	-----------------------	-----

Daftar Tabel

2.1	Operator pada Python	22
3.1	Operator-operator perbandingan (comparison).	36
3.2	Operator-operator perbandingan (comparison).	37
3.3	Contoh input dan output yang diharapkan.	43
14.1	Special Character pada Python	160
14.2	Escaped Character pada Regex	160
14.3	Himpunan Karakter pada Regex	161
14.4	Fungsi Regex pada Python	161

Pengenalan Bahasa Pemrograman Python

1	Bahasa Pemrograman Python	1
1.1	Tujuan Praktikum	
1.2	Alat dan Bahan	
1.3	Materi	
1.3.1	Kenapa Python?	
1.3.2	Menginstall Python 3	
1.3.3	Menjalankan Python Mode Interaktif	
1.3.4	Editor untuk Python	
1.3.5	Menjalankan Script Python di Terminal/Console	
1.3.6	Mencari Bug dan Memperbaikinya (debugging)	
1.4	Kegiatan Praktikum	
1.4.1	Menginstall Package Jupyter Notebook	
1.4.2	Eksplorasi Python mode Interaktif	
1.5	Latihan Mandiri	
2	Variable, Expression dan Statements	19
2.1	Tujuan Praktikum	
2.2	Alat dan Bahan	
2.3	Materi	
2.3.1	Values dan type	
2.3.2	Variabel	
2.3.3	Nama Variabel dan Keywords	
2.3.4	Statements	
2.3.5	Operator dan Operand	
2.3.6	Expressions	
2.3.7	Urutan Operasi	
2.3.8	Operator Modulus dan String	
2.3.9	Menangani Input dari Pengguna	
2.4	Komentar	
2.5	Kegiatan Praktikum	
2.5.1	Membuat Variabel	
2.5.2	Memberikan nilai dalam variabel	
2.5.3	Mencetak nilai dalam variabel	
2.5.4	Separator, tipe data dan fungsi type	
2.6	Latihan Mandiri	

1. Bahasa Pemrograman Python

1.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menginstall Python pada sistem operasi yang digunakan.
2. Dapat menjalankan perintah-perintah Python dalam mode interaktif.
3. Mengenal beberapa pilihan editor untuk Python dan dapat menggunakan salah satunya untuk menjalankan script Python.
4. Dapat menjalankan script Python di Terminal/Console maupun di Web (dengan Jupyter Notebook).
5. Dapat menginstall maupun mengupdate package.
6. Dapat menemukan kesalahan pada script Python berdasarkan informasi dari interpreter Python.

1.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

1.3 Materi

1.3.1 Kenapa Python?

Python adalah bahasa pemrograman level tinggi yang *interpreted*, mendukung Object Oriented Programming (OOP) dan memiliki sifat *dynamic semantics*. Menurut sebuah survey yang diselenggarakan oleh Stackoverflow (<https://insights.stackoverflow.com/survey/2019#technology>), Python merupakan salah satu bahasa pemrograman yang paling banyak dipakai setelah Javascript. Secara umum Python merupakan salah satu bahasa pemrograman yang paling populer di dunia karena aturan dan sintaksnya yang sederhana untuk dipelajari bagi pemula. Logo dari Python dapat dilihat pada Gambar 1.1.



Gambar 1.1: Logo Python (diambil dari <https://www.python.org/>)

Kesederhanaan bentuk dan sintaks Python dapat dilihat dari perbandingan source code program Hello World jika dibandingkan dengan Java dan C.

Pada bahasa pemrograman Java:

```
1 public class Main {  
2     public static void main(String[] args){  
3         System.out.println("Hello World!");  
4     }  
5 }
```

Pada bahasa pemrograman C:

```
1 #include<stdio.h>  
2  
3 int main(){  
4     printf("Hello World");  
5     return 0;  
6 }
```

Pada bahasa pemrograman Python:

```
1 print("Hello World!");
```

Beberapa kelebihan dari bahasa pemrograman Python antara lain:

- Dukungan pustaka pihak ketiga yang sangat kaya dan beragam. Pustaka-pustaka tersebut menyebabkan Python dapat digunakan untuk menyelesaikan masalah di berbagai macam bidang. Sebagai contoh Python banyak digunakan di bidang data science karena ada banyak sekali pustaka-pustaka pendukung seperti pandas, numpy, tensorflow, keras dan library-library machine learning lainnya.

- Pustaka bawaan dari Python sendiri sudah sangat beragam dan mencakup banyak sekali aspek-aspek dasar yang umumnya dibutuhkan oleh programmer, antara lain dukungan terhadap basis data, pengaksesan jaringan, pengaksesan fitur-fitur di sistem operasi dan masih banyak lainnya.
- Python memiliki lisensi Open Source sehingga dapat digunakan secara bebas bahkan dapat digunakan untuk keperluan komersial tanpa perlu membayar lisensi. Selain itu Python juga aktif dikembangkan sehingga fitur-fitur dan kemampuannya selalu bertambah.
- Python relatif mudah dipelajari untuk pemula karena sintaksnya yang sederhana dan sangat menyerupai bahasa Inggris. Selain itu juga tersedia banyak dokumentasi, tutorial dan bahkan online course tentang Python yang memudahkan seorang pemula untuk mempelajarinya.

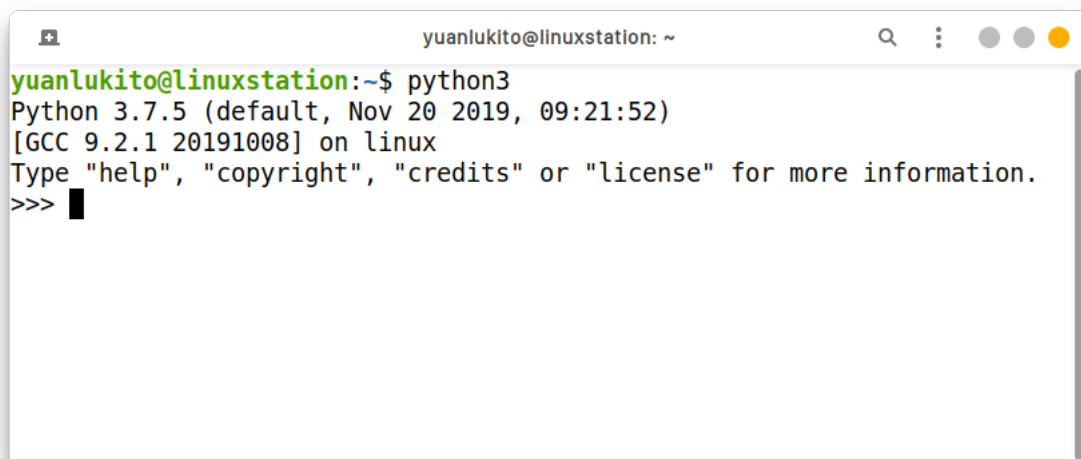
Walaupun memiliki banyak kelebihan, Python juga memiliki beberapa kekurangan sebagai berikut:

- Saat ini Python belum mendukung untuk pembuatan aplikasi di platform mobile seperti Android atau iOS.
- Konsumsi memory yang relatif besar sehingga tidak cocok digunakan untuk kasus-kasus yang membutuhkan memory dalam jumlah sangat besar.
- Kecepatan proses dari Python relatif lebih lambat jika dibandingkan dengan bahasa pemrograman seperti C.

Sebagai pemula, anda tidak perlu terlalu memikirkan tentang kelemahan-kelemahan Python karena kemungkinan besar anda tidak akan menjumpai masalah-masalah tersebut. Dengan bahasa pemrograman Python, diharapkan anda dapat belajar algoritma dan pemrograman dengan lebih cepat dan lebih mudah jika dibandingkan menggunakan bahasa pemrograman lainnya.

1.3.2 Menginstall Python 3

Untuk saat ini terdapat dua macam versi Python yang banyak digunakan, yaitu Python versi 2 dan versi 3. Untuk keperluan mata kuliah ini digunakan Python 3 yang merupakan versi yang lebih baru. Bagi anda yang menggunakan sistem operasi Linux maupun macOS, biasanya Python versi 3 sudah terinstall dan siap digunakan. Pada Ubuntu, Python dapat dijalankan pada terminal menggunakan perintah **python3** seperti yang ditunjukkan pada Gambar 1.2.



```
yuanlukito@linuxstation: ~$ python3
Python 3.7.5 (default, Nov 20 2019, 09:21:52)
[GCC 9.2.1 20191008] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Gambar 1.2: Python versi 3 di Ubuntu Linux. Terinstall Python versi 3.7.5

Bagi anda yang menggunakan Windows, anda bisa menginstall distribusi Anaconda yang relatif mudah untuk diinstall. Pilihlah Anaconda Individual Edition 64-Bit Graphical Installer, seperti

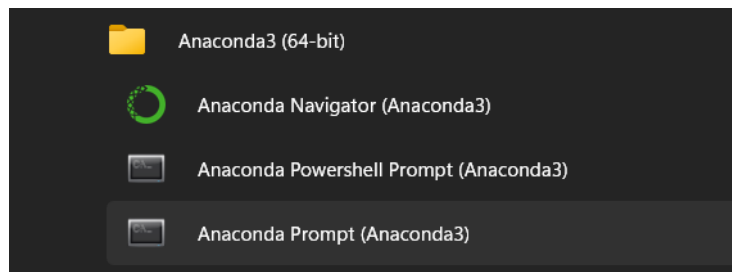
pada Gambar 1.3. Pada saat modul ini dibuat, versi yang bisa di-download adalah Python 3.9. Sesuaikan dengan sistem operasi yang sedang anda gunakan.



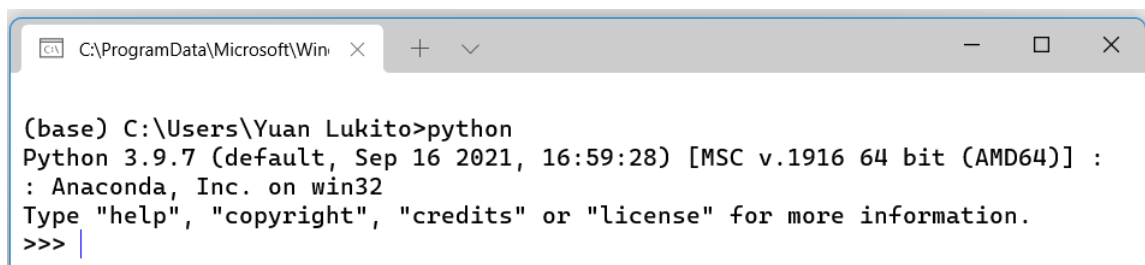
Gambar 1.3: Distribusi Anaconda Individual Edition.

1.3.3 Menjalankan Python Mode Interaktif

Untuk masuk mode interaktif, jalankan perintah **python3** pada terminal di Ubuntu. Jika anda menggunakan Windows, jalankan Anaconda Prompt, kemudian ketikkan perintah **python** seperti yang ditunjukkan pada Gambar 1.4 dan Gambar 1.5.



Gambar 1.4: Menu Anaconda Prompt setelah terinstall di Windows.



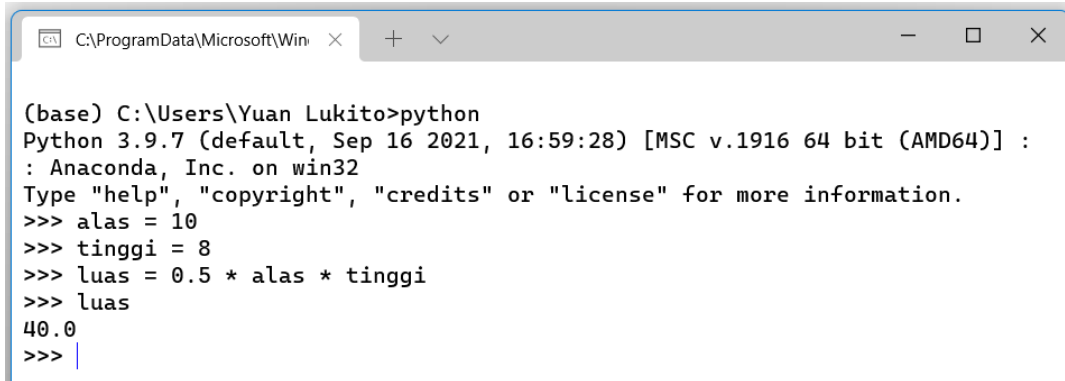
Gambar 1.5: Informasi versi Python akan ditampilkan dan siap menerima perintah.

Penggunaan mode interaktif ini memungkinkan anda memasukkan perintah satu-persatu dan langsung diproses oleh interpreter Python. Sebagai contoh, hitunglah luas segitiga yang alasnya memiliki panjang 10 cm dan tinggi 8 cm. Untuk menghitung luas segitiga tersebut, ketikkan

masing-masing perintah berikut ini secara berurutan baris-perbaris, diakhiri dengan menekan tombol Enter.

- `alas = 10`
- `tinggi = 8`
- `luas = 0.5 * alas * tinggi`
- `luas`

Hasilnya akan seperti yang ditunjukkan pada Gambar 1.6.

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\ProgramData\Microsoft\Win...'. The prompt is '(base) C:\Users\Yuan Lukito>python'. The output shows 'Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] : Anaconda, Inc. on win32'. It then shows the execution of the code: '>>> alas = 10', '>>> tinggi = 8', '>>> luas = 0.5 * alas * tinggi', and '>>> luas' resulting in '40.0'. The prompt '>>>' is followed by a vertical bar cursor.

```
(base) C:\Users\Yuan Lukito>python
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] :
: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> alas = 10
>>> tinggi = 8
>>> luas = 0.5 * alas * tinggi
>>> luas
40.0
>>> |
```

Gambar 1.6: Menghitung Luas Segitiga dengan bantuan Python

Perintah pertama dan kedua adalah perintah *assignment*, yaitu mengisi nilai variabel panjang dengan nilai 10 dan variabel tinggi diisi dengan nilai 8. Sedangkan perintah ketiga mengisi variabel luas dengan hasil perhitungan $0.5 * panjang * tinggi$. Perintah terakhir adalah menampilkan isi dari variabel luas.

Variabel adalah tempat kita menyimpan nilai yang nantinya akan dipakai untuk proses-proses berikutnya. Pembahasan variabel akan dilakukan pada Bab 2.

Untuk mengakhiri sesi interaktif, ketikkan perintah **exit()** yang diakhiri dengan tombol Enter.

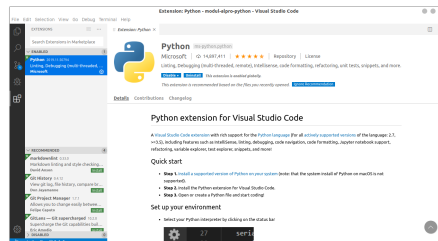
1.3.4 Editor untuk Python

Untuk menulis suatu program, anda memerlukan editor untuk menuliskan source code yang diinginkan. Semakin canggih suatu editor, biasanya akan sangat mempermudah dalam pembuatan program. Beberapa editor yang biasanya dipakai untuk membuat program dalam bahasa pemrograman Python antara lain:

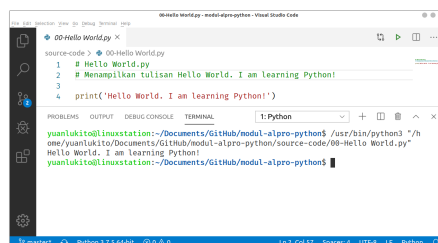
- Visual Studio Code + Python Extension for Visual Studio Code (<https://code.visualstudio.com/>).
- PyCharm (<https://www.jetbrains.com/pycharm/>).
- Spyder (<https://www.spyder-ide.org/>).
- ActivePython (<https://www.activestate.com/products/activepython/>).
- IDLE (<https://docs.python.org/3/library/idle.html>).

Sebenarnya PyCharm dan Spyder tidak hanya berupa editor saja, melainkan sebuah Integrated Development Environment. Sebuah aplikasi yang dapat digunakan untuk mengembangkan aplikasi-aplikasi yang seluruh fasilitasnya sudah tersedia dan terintegrasi. Biasanya IDE dipakai untuk project-project dengan skala menengah ke atas. Untuk keperluan praktikum, penggunaan editor sudah dianggap cukup dengan pertimbangan kesederhanaan tampilan dan fasilitas yang disediakan sudah mencukupi untuk kebutuhan selama praktikum.

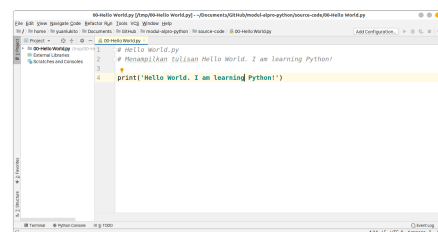
Contoh tampilan Visual Studio Code yang sudah terpasang Python Extension dapat dilihat pada Gambar 1.7 dan Gambar 1.8. Editor lain seperti PyCharm punya fitur yang lebih lengkap dan dapat digunakan untuk berbagai macam bentuk project Python. Tampilan PyCharm dapat dilihat pada Gambar 1.9. Modul ini menggunakan Visual Studio Code sebagai editor utama dan Spyder/PyCharm sebagai editor penunjang. Anda lebih baik menginstall semuanya sekaligus untuk mempelajari kelebihan dan kekurangannya masing-masing.



Gambar 1.7: Python Extension untuk Visual Studio Code.



Gambar 1.8: Tampilan Visual Studio Code saat menjalankan script Python.



Gambar 1.9: Tampilan PyCharm saat mengedit file Python.

1.3.5 Menjalankan Script Python di Terminal/Console

Anda sebelumnya sudah mencoba menjalankan Python dalam mode interaktif (seperti pada Gambar 1.6). Pada mode interaktif anda harus mengetik perintah-perintah yang diinginkan satu-persatu. Setiap perintah yang anda masukkan langsung dijalankan oleh interpreter Python. Mode interaktif ini cocok digunakan untuk mencoba-coba fungsi atau perintah-perintah baru karena hasilnya akan langsung didapatkan. Kekurangan dari mode interaktif adalah anda harus mengetik perintahnya satu-persatu dan harus mengulanginya lagi jika anda sudah terlanjur keluar dari mode interaktif.

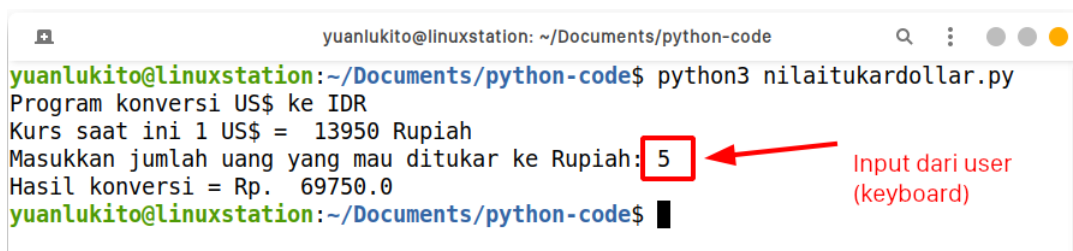
Interpreter Python juga mendukung mode script, yaitu menjalankan daftar perintah-perintah (yang disimpan dalam file .py) secara langsung tanpa anda harus mengetikkannya lagi satu-persatu. Untuk menjalankan script Python menggunakan Terminal/Command Prompt, bukalah Terminal

(pada Ubuntu) atau Anaconda Prompt (pada Windows). Ketikkan perintah **python3 namafile.py** (Ubuntu) atau **python namafile.py** pada Windows.

Untuk mencoba, buatlah sebuah file baru bernama **nilaitukardollar.py**, ketiklah kode program berikut di dalam file tersebut, kemudian simpan di suatu tempat.

```
1  # nilai kurs 1 US$ ke IDR
2  kursUSD = 13950
3
4  # informasi program
5  print('Program konversi US$ ke IDR')
6  print('Kurs saat ini 1 US$ = ',kursUSD, 'Rupiah')
7  # input jumlah US$ yang mau ditukar
8  jumlahUSD = float(input('Masukkan jumlah uang yang mau ditukar ke Rupiah: '))
9  # hitung nilainya dalam Rupiah
10 dalamRupiah = jumlahUSD * kursUSD
11 # tampilkan hasilnya
12 print('Hasil konversi = Rp. ', dalamRupiah)
13
```

Untuk menjalankan script tersebut, gunakan perintah **python3 nilaitukardollar.py** (Ubuntu) atau **python nilaitukardollar.py** (Windows). Pada Ubuntu, tampilan hasilnya dapat dilihat pada Gambar 1.10.



```
yuanlukito@linuxstation: ~/Documents/python-code
yuanlukito@linuxstation:~/Documents/python-code$ python3 nilaitukardollar.py
Program konversi US$ ke IDR
Kurs saat ini 1 US$ = 13950 Rupiah
Masukkan jumlah uang yang mau ditukar ke Rupiah: 5
Hasil konversi = Rp. 69750.0
yuanlukito@linuxstation:~/Documents/python-code$
```

Gambar 1.10: Menjalankan script Python di Terminal Ubuntu.

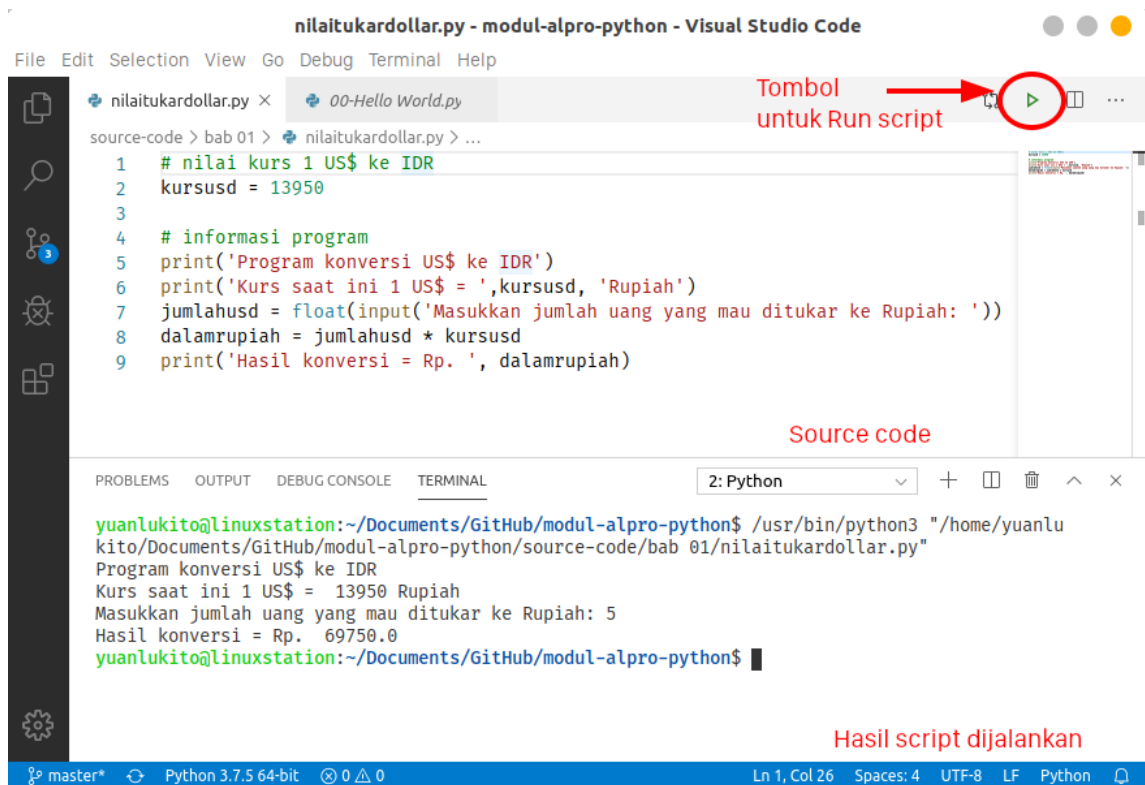
Baris yang diawali dengan tanda # adalah baris komentar. Komentar digunakan untuk memberikan informasi atau keterangan mengenai baris-baris program dan akan diabaikan oleh interpreter Python.

Jika anda menggunakan editor seperti Visual Studio Code, anda tidak perlu repot menjalankan terminal dan mengetik perintahnya. Pada Visual Studio Code sudah ada tombol Run untuk menjalankan script tersebut. Anda juga dapat berinteraksi dengan hasil program melalui terminal yang disediakan. Tampilan Visual Studio Code saat menjalankan script **nilaitukardollar.py** dapat dilihat pada Gambar 1.11.

1.3.6 Mencari Bug dan Memperbaikinya (debugging)

Bug merupakan istilah dalam pemrograman yang berarti ada kesalahan dalam program. Kesalahan dalam membuat program secara umum dapat digolongkan menjadi dua macam, yaitu:

- Kesalahan dalam menulis program. Biasa disebut sebagai *syntax error* atau *compile error*. Kesalahan jenis ini biasanya diakibatkan oleh kesalahan dalam mengetik (typo). Jenis



Gambar 1.11: Menjalankan script Python pada Visual Studio Code.

kesalahan ini mudah sekali untuk ditemukan dengan bantuan interpreter Python maupun editor yang dipakai.

- Kesalahan saat program berjalan. Biasa disebut sebagai *runtime error*. Jenis kesalahan ini lebih sulit ditemukan dan diperbaiki dibanding jenis kesalahan yang pertama.

Python merupakan bahasa pemrograman yang menggunakan sistem interpreter, artinya kode program anda akan dibaca baris-perbaris untuk dicek apakah ada kesalahan dalam penulisan perintah. Jika baris yang sedang dicek memenuhi aturan penulisan program (sintaks), maka baris tersebut langsung dijalankan. Jika terjadi kesalahan pada baris tersebut saat dijalankan, program Python anda akan langsung berhenti dan tidak melanjutkan ke baris berikutnya. Sebagai contoh, perhatikan kode program berikut ini:

```

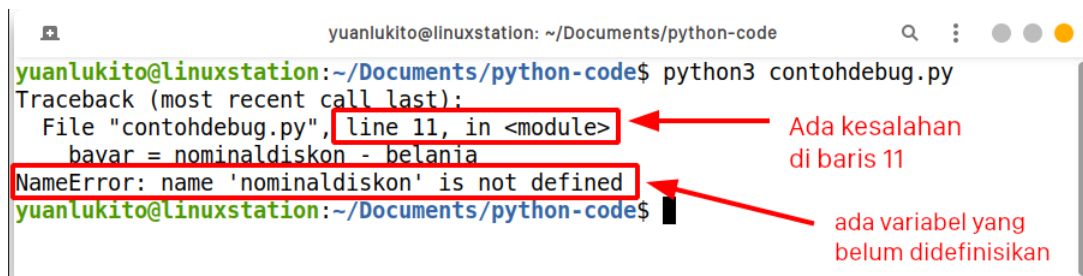
1  #jumlah belanja (dalam rupiah)
2  belanja = 100000
3
4  #besarnya diskon (dalam persen)
5  diskon = 30
6
7  #besarnya diskon
8  nominal_diskon = (diskon/100) * belanja
9
10 #hitung jumlah yang harus dibayar
11 bayar = nominal_diskon - belanja
12
13 #tampilkan hasilnya

```

14 `print('Anda harus membayar: Rp. ', bayar)`

Program tersebut menghitung jumlah yang harus dibayarkan jika belanja sebesar Rp. 100.000 dan mendapat diskon 30%. Jika script tersebut dijalankan, hasilnya akan seperti pada Gambar 1.12. Saat script dijalankan, tidak ada hasil yang dikeluarkan karena terjadi kesalahan pada baris ke-11. Ada keterangan jenis kesalahan, yaitu **NameError**. Kesalahan ini merupakan kesalahan sintaks karena ada aturan penulisan yang tidak terpenuhi. Kesalahan terjadi pada variabel bernama **nominaldiskon** yang belum didefinisikan tetapi nilainya sudah digunakan.

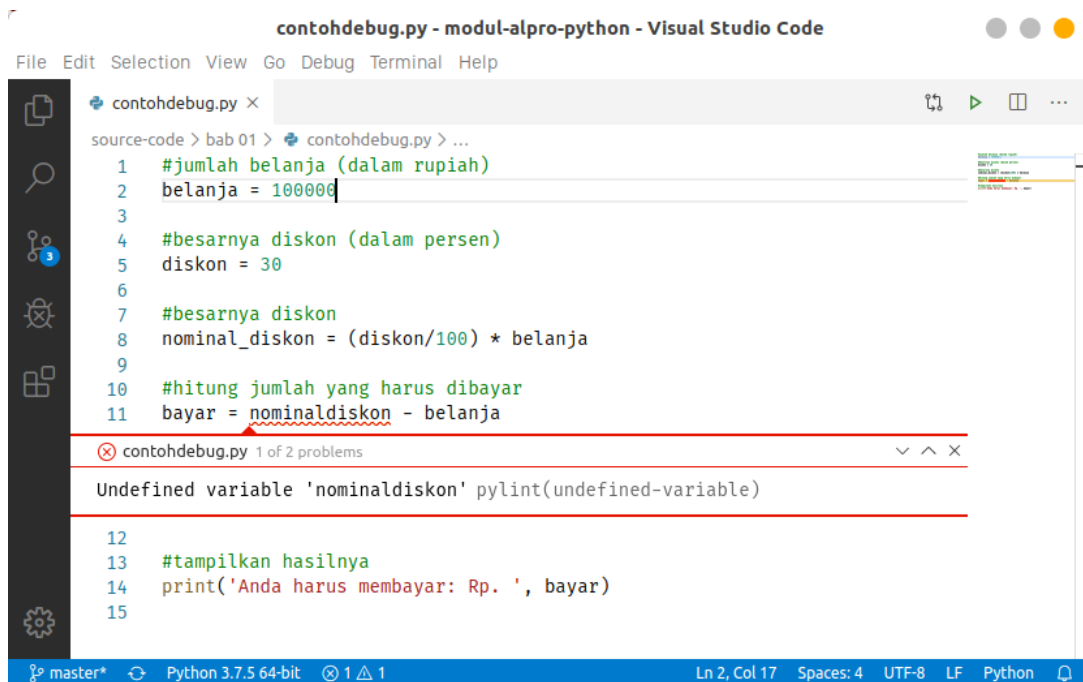
Jika anda perhatikan kode programnya, pada baris ke-8 ada variabel bernama **nominal_diskon**, tetapi pada baris ke-11 dituliskan sebagai **nominaldiskon**. Terjadi kesalahan pengetikan (typo) yang menyebabkan script tidak bisa dijalankan.



```
yuanlukito@linuxstation: ~/Documents/python-code
yuanlukito@linuxstation:~/Documents/python-code$ python3 contohdebug.py
Traceback (most recent call last):
  File "contohdebug.py", line 11, in <module>
    bayar = nominaldiskon - belanja
NameError: name 'nominaldiskon' is not defined
yuanlukito@linuxstation:~/Documents/python-code$
```

Gambar 1.12: Kesalahan yang muncul saat script dijalankan.

Jika anda menggunakan editor yang mendukung Python, biasanya sebelum dijalankan sudah akan dilakukan pengecekan dan apabila ada kesalahan sintaks dapat diketahui lebih awal. Gambar 1.13 adalah contoh tampilan di Visual Studio Code untuk script tersebut. Untuk memperbaiki kesalahan tersebut, ganti baris ke-8 menjadi **nominaldiskon**, seperti pada kode program berikut ini:



```
contohdebug.py - modul-alpro-python - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
contohdebug.py x
source-code > bab 01 > contohdebug.py > ...
1 #jumlah belanja (dalam rupiah)
2 belanja = 100000
3
4 #besarnya diskon (dalam persen)
5 diskon = 30
6
7 #besarnya diskon
8 nominal_diskon = (diskon/100) * belanja
9
10 #hitung jumlah yang harus dibayar
11 bayar = nominaldiskon - belanja
12
13 #tampilkan hasilnya
14 print('Anda harus membayar: Rp. ', bayar)
15

x contohdebug.py 1 of 2 problems
Undefined variable 'nominaldiskon' pylint(undefined-variable)

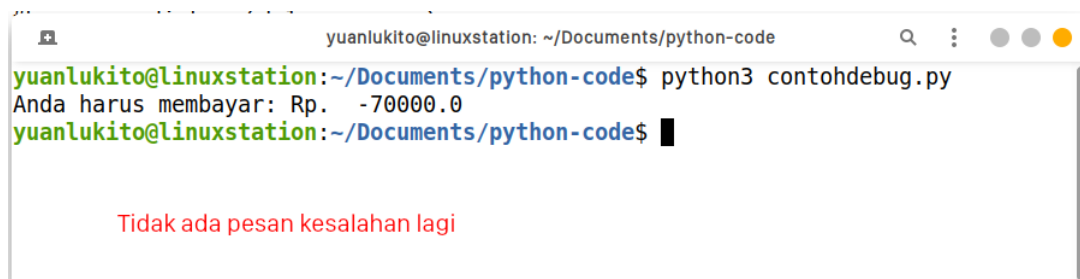
12
13 #tampilkan hasilnya
14 print('Anda harus membayar: Rp. ', bayar)
15

master Python 3.7.5 64-bit 1 Ln 2, Col 17 Spaces: 4 UTF-8 LF Python
```

Gambar 1.13: Visual Studio Code menemukan kesalahan sintaks sebelum script dijalankan.

```
1  #jumlah belanja (dalam rupiah)
2  belanja = 100000
3
4  #besarnya diskon (dalam persen)
5  diskon = 30
6
7  #besarnya diskon
8  nominaldiskon = (diskon/100) * belanja
9
10 #hitung jumlah yang harus dibayar
11 bayar = nominaldiskon - belanja
12
13 #tampilkan hasilnya
14 print('Anda harus membayar: Rp. ', bayar)
```

Setelah diperbaiki, maka sekarang script dapat dijalankan dengan baik sampai selesai seperti yang terlihat pada Gambar 1.14.



```
yuanlukito@linuxstation: ~/Documents/python-code
yuanlukito@linuxstation:~/Documents/python-code$ python3 contohdebug.py
Anda harus membayar: Rp. -70000.0
yuanlukito@linuxstation:~/Documents/python-code$
```

Tidak ada pesan kesalahan lagi

Gambar 1.14: Script sudah bisa dijalankan sampai selesai dengan baik.

Jika anda perhatikan dengan baik, script tersebut berjalan dijalankan dengan baik sampai selesai, tetapi hasilnya tidak sesuai dengan yang diharapkan. Jika kita membeli barang Rp. 100.000 dan mendapat diskon 30%, maka seharusnya yang harus dibayarkan adalah Rp. 70.000. Tetapi script tersebut menghasilkan output -70000. Kesalahan ini adalah kesalahan jenis **runtime error**, yaitu kesalahan yang muncul saat program dijalankan. Kesalahan jenis ini biasanya diakibatkan oleh kesalahan algoritma, kesalahan logika atau kesalahan-kesalahan dalam mengatur langkah-langkah dalam program.

Kesalahan terjadi pada baris ke-11, yaitu pada bagian perhitungan jumlah yang harus dibayar, seharusnya jumlah yang harus dibayar adalah harga barang dikurangi diskon. Pada script tersebut perhitungannya terbalik sehingga menjadi salah. Perbaiki script tersebut menjadi seperti berikut ini:

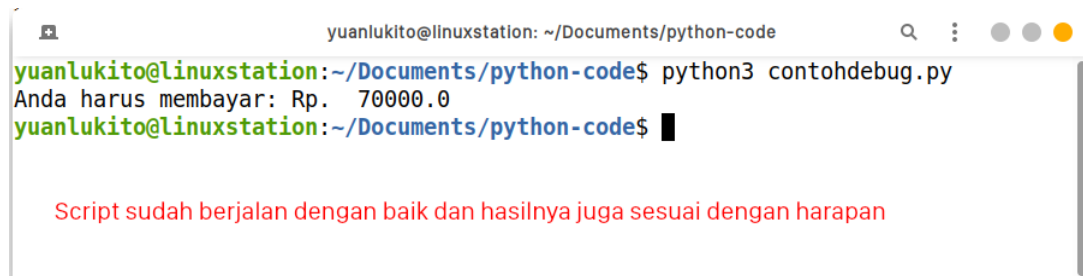
```
1  #jumlah belanja (dalam rupiah)
2  belanja = 100000
3
4  #besarnya diskon (dalam persen)
5  diskon = 30
6
7  #besarnya diskon
8  nominaldiskon = (diskon/100) * belanja
9
```

```

10 #hitung jumlah yang harus dibayar
11 # INI SALAH: bayar = nominaldiskon - belanja
12 bayar = belanja - nominaldiskon
13
14 #tampilkan hasilnya
15 print('Anda harus membayar: Rp. ', bayar)

```

Setelah diperbaiki, script tersebut akan menghasilkan hasil yang benar sesuai dengan yang diharapkan. Hasilnya dapat dilihat pada Gambar 1.15.



```

yuanlukito@linuxstation: ~/Documents/python-code
yuanlukito@linuxstation:~/Documents/python-code$ python3 contohdebug.py
Anda harus membayar: Rp. 70000.0
yuanlukito@linuxstation:~/Documents/python-code$

```

Script sudah berjalan dengan baik dan hasilnya juga sesuai dengan harapan

Gambar 1.15: Hasil sudah sesuai dengan yang diharapkan.

1.4 Kegiatan Praktikum

Kegiatan praktikum yang akan dilakukan adalah sebagai berikut:

1. Menginstall Package Jupyter Notebook dan menjalankan script menggunakan Jupyter Notebook.
2. Eksplorasi Python mode interaktif untuk perhitungan-perhitungan sederhana.

1.4.1 Menginstall Package Jupyter Notebook

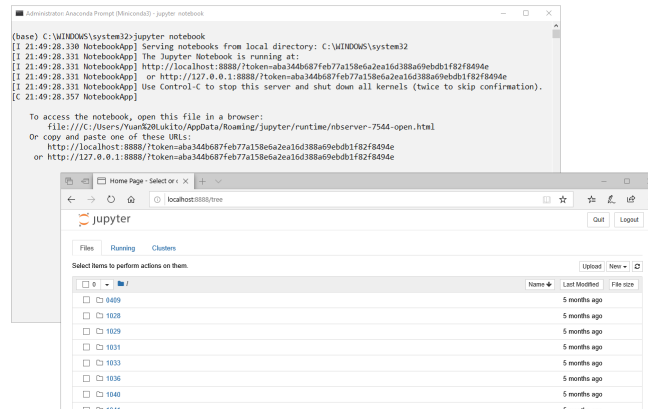
Untuk menginstall package Python kita akan menggunakan perintah **pip**. Pip merupakan singkatan dari **package installer for Python**. Pada bagian ini dibahas mengenai instalasi package Jupyter Notebook. Jupyter Notebook adalah aplikasi web yang dapat bertindak sebagai editor Python, menampilkan visualisasi data maupun menampilkan hasil dari menjalankan script Python. Jupyter Notebook banyak dipakai karena hanya dibutuhkan browser untuk mengaksesnya. Tampilan Jupyter Notebook dapat dilihat pada Gambar 1.16.

Untuk menginstall Jupyter Notebook, jalankan Anaconda Prompt (atau Terminal bila anda menggunakan Linux). Kemudian ketik perintah **pip install jupyter notebook**. Pada Ubuntu, perintah yang digunakan adalah **pip3 install jupyter notebook**. Jika anda mengetikkan perintah tersebut dengan benar, pip akan mendownload package-package yang diperlukan dan sekaligus menginstallnya. Tampilan pip saat dijalankan dapat dilihat pada Gambar 1.17.

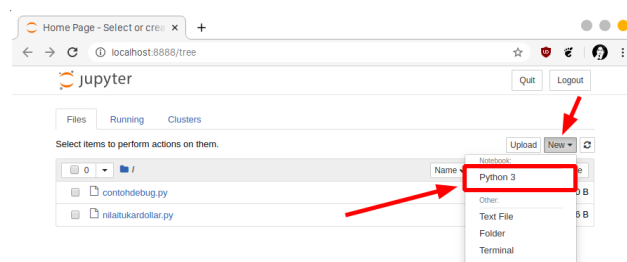
Untuk instalasi package gunakan perintah **pip install <nama package>**

Setelah proses instalasi berhasil, untuk menjalankannya ketikkan perintah **jupyter notebook**. Biasanya akan terbuka browser yang mengarah langsung ke halaman pertama Jupyter Notebook, seperti yang terlihat pada Gambar 1.18.

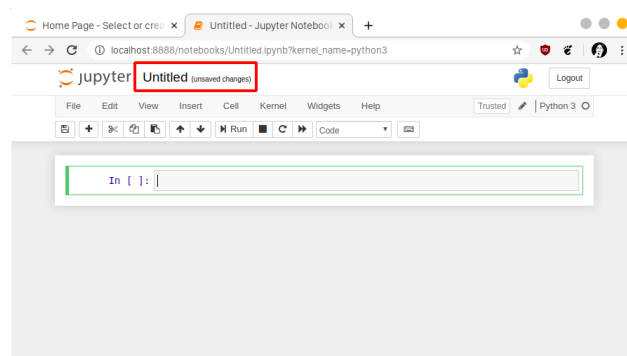
Untuk menjalankan Jupyter Notebook, gunakan perintah **jupyter notebook** pada Anaconda Prompt atau Terminal.



Gambar 1.18: Tampilan awal Jupyter Notebook.



Gambar 1.19: Menu untuk membuat notebook baru.



Gambar 1.20: Tampilan awal notebook baru di Jupyter Notebook.

1.4.2 Eksplorasi Python mode Interaktif

Pada bagian ini kita akan mencoba menggunakan Python metode interaktif untuk menjawab beberapa pertanyaan-pertanyaan berikut ini:

■ Contoh 1.1 Menghitung Usia

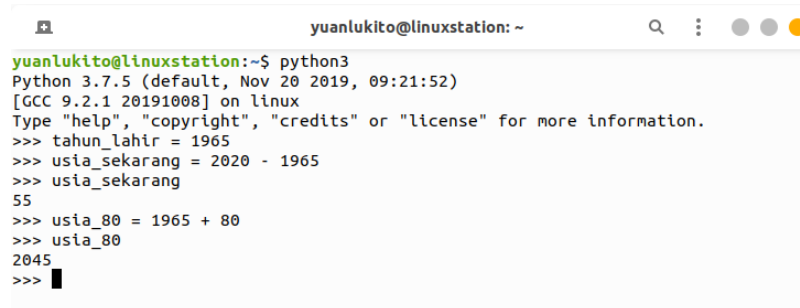
Jika diketahui tahun lahir seseorang, kita bisa menentukan usianya sekarang. Misalnya Andi lahir tahun 1990 dan sekarang adalah tahun 2020, maka usianya adalah 30 tahun. Gunakan Python untuk menjawab beberapa masalah berikut ini (diasumsikan sekarang adalah tahun 2020):

- Trump dilahirkan tahun 1965. Berapa usianya sekarang?
- Pada tahun berapa Trump akan berusia 80 tahun?

■ Untuk menjawab pertanyaan-pertanyaan tersebut, langkah-langkahnya adalah sebagai berikut:

1. Tahun lahir = 1965.
2. Usia sekarang (2020) = $2020 - 1965 = 55$ tahun.
3. Usia Trump 80 saat = $1965 + 80 = 2045$.

Langkah-langkah penyelesaiannya dalam Python dapat dilihat pada Gambar 1.21.



```
yuanlukito@linuxstation: ~  
yuanlukito@linuxstation:~$ python3  
Python 3.7.5 (default, Nov 20 2019, 09:21:52)  
[GCC 9.2.1 20191008] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> tahun_lahir = 1965  
>>> usia_sekarang = 2020 - 1965  
>>> usia_sekarang  
55  
>>> usia_80 = 1965 + 80  
>>> usia_80  
2045  
>>> █
```

Gambar 1.21: Penyelesaian untuk masalah usia Trump.

■ Contoh 1.2 Compound Interest

Compound interest dalam bahasa Indonesia adalah "Bunga ber-bunga". Istilah ini biasanya digunakan dalam dunia keuangan, misalnya perbankan, asuransi maupun investasi. Suatu Bank bernama **Bank Bank Toet** menyediakan produk deposito bernama **Pasti Cuan** dengan bunga 10% per-tahun. Artinya jika anda menyimpan uang anda di deposito tersebut, uang anda akan bertambah sebanyak 10% setiap tahunnya. Gunakan Python untuk mencari jawaban dari contoh masalah berikut ini:

- Jika setoran awal sebanyak Rp. 5 juta rupiah dan disimpan selama 3 tahun tanpa pernah diambil, berapa jumlah uang di akhir tahun ketiga?

■ Untuk menyelesaikan masalah yang pertama, langkah-langkah yang perlu dilakukan adalah sebagai berikut:

1. Saldo awal Rp. 5 juta.
2. Di akhir tahun pertama, bunga yang didapat adalah $10\% * 5.000.000 = 500.000$.
3. Sehingga saldo di akhir tahun pertama adalah $5.000.000 + 500.000 = 5.500.000$.
4. Di akhir tahun kedua, bunga yang didapat adalah $10\% * 5.500.000 = 550.000$. Ini yang dinamakan sebagai compound interest.
5. Sehingga saldo di akhir tahun kedua adalah $5.500.000 + 550.000 = 6.050.000$.
6. Di akhir tahun ketiga, bunga yang didapat adalah $10\% * 6.050.000 = 605.000$.
7. Sehingga saldo di akhir tahun ketiga adalah $6.050.000 + 605.000 = 6.655.000$.

Setelah anda memahami permasalahan dan dapat menyusun langkah-langkah penyelesaiannya, barulah anda dapat mulai menuliskan programnya menggunakan Python.

Jalankan Python mode interaktif, kemudian masukkan perintah-perintah seperti pada Gambar 1.22.

■ Contoh 1.3 Formula Compound Interest

Perhitungan compound interest pada contoh sebelumnya menghitung saldo di akhir tahun ketiga. Bagaimana jika dibutuhkan menghitung berapa saldo di akhir tahun ke-dua puluh? Tentunya akan sangat melelahkan memasukkan perintah-perintah tersebut secara berulang-ulang untuk menghitung simpanan selama dua puluh tahun. Cara lain yang dapat kita gunakan adalah menggunakan


```
yuanlukito@linuxstation: ~  
yuanlukito@linuxstation:~$ python3  
Python 3.7.5 (default, Nov 20 2019, 09:21:52)  
[GCC 9.2.1 20191008] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> saldo_awal = 5000000  
>>> bunga_tahun_satu = 0.1 * saldo_awal  
>>> saldo_tahun_satu = saldo_awal + bunga_tahun_satu Tahun 1  
>>> saldo_tahun_satu  
5500000.0  
>>> bunga_tahun_dua = 0.1 * saldo_tahun_satu  
>>> saldo_tahun_dua = saldo_tahun_satu + bunga_tahun_dua Tahun 2  
>>> saldo_tahun_dua  
6050000.0  
>>> bunga_tahun_tiga = 0.1 * saldo_tahun_dua  
>>> saldo_akhir = saldo_tahun_dua + bunga_tahun_tiga Tahun 3  
>>> saldo_akhir  
6655000.0  
>>> █  
  
Saldo akhir adalah 6.655.000
```

Gambar 1.22: Hasil perhitungan compound interest selama tiga tahun.

formula compound interest seperti berikut ini:

$$A = P\left(1 + \frac{r}{n}\right)^{nt} \quad (1.1)$$

- A = saldo akhir.
- P = saldo awal.
- r = besarnya bunga.
- n = jumlah perhitungan bunga yang dilakukan (dalam 1 tahun berapa kali).
- t = jumlah periode.

Tuliskan formula dan perhitungannya menggunakan Python! █

Dengan adanya rumus tersebut, langkah-langkah yang diperlukan akan lebih singkat, yaitu:

1. Tentukan saldo awal (P = 5.000.000).
2. Tentukan besarnya bunga (r = 10% = 0.1).
3. Tentukan berapa kali bunga didapatkan (n = 1, karena bunga dihitung satu kali per-tahun).
4. Tentukan periodenya (t = 3 tahun).
5. Masukkan dalam formula compound interest, kita dapatkan saldo akhirnya.

Secara umum dalam membuat program anda perlu terlebih dahulu menentukan langkah-langkah yang diperlukan untuk memecahkan masalah. Jangan terburu-buru untuk langsung membuat programnya. Perhitungan dengan formula compound interest dapat anda lihat pada Gambar 1.23.

```
yuanlukito@linuxstation: ~  
yuanlukito@linuxstation:~$ python3  
Python 3.7.5 (default, Nov 20 2019, 09:21:52)  
[GCC 9.2.1 20191008] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> P = 5000000  
>>> r = 0.1  
>>> n = 1  
>>> t = 3  
>>> A = P * (1 + r/n) ** (n*t)  
>>> A  
6655000.000000002  
>>> █  
  
Hasilnya 6.655.000  
sama seperti hasil sebelumnya
```

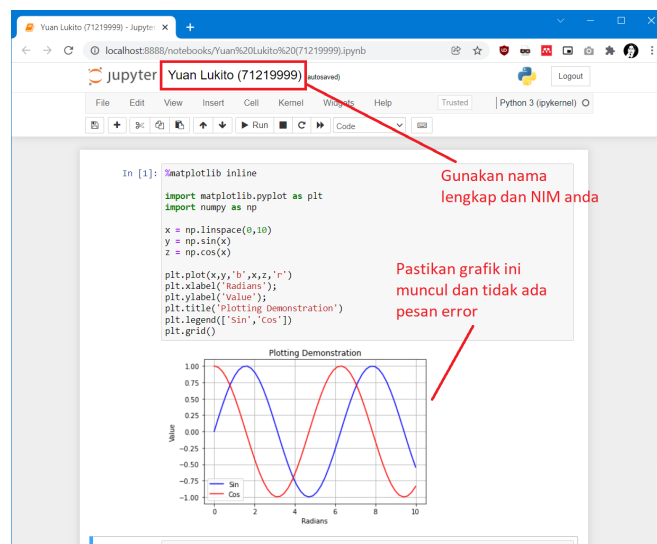
Gambar 1.23: Hasil perhitungan compound interest menggunakan formula.

- # Sebelum membuat program dalam Python, pastikan anda sudah menyusun langkah-langkah penyelesaiannya terlebih dahulu. Langkah-langkah penyelesaian suatu masalah biasa disebut sebagai **algoritma**.

1.5 Latihan Mandiri

Latihan 1.1 Buatlah satu Notebook baru, berilah nama notebook tersebut dengan format **Nama lengkap (NIM)** anda. Kemudian ketikkan ulang kode program seperti yang ada di Gambar 1.24.

Tuliskan langkah-langkah yang anda lakukan secara berurutan, sampai mendapatkan hasil seperti pada gambar tersebut. Perhatikan, anda harus menjelaskan langkah-langkah disertai gambar/screenshot. Jangan lupa memasukkan langkah-langkah instalasi package-package yang diperlukan (matplotlib dan numpy). Instalasi package tersebut biasanya tidak diperlukan jika anda menginstall Python dari Anaconda.



Gambar 1.24: Menggunakan Jupyter Notebook untuk menghasilkan grafik.

Latihan 1.2 Gerard membeli emas 25 gram dengan harga Rp. 650.000/gram. Jika sekarang harga emas menjadi Rp. 685.000/gram,

- Berapa keuntungan yang didapat oleh Gerard (dalam Rp dan dalam %)?
- Jika Gerard kemudian membeli lagi 15 gram emas dengan harga Rp. 685.000, maka Gerard sekarang memiliki total 40 gram emas. Jika kemudian harga emas naik lagi menjadi Rp. 715.000, berapa keuntungan yang didapat oleh Gerard (dalam Rp dan dalam %)?

Gunakan Python mode interaktif untuk menyelesaikan permasalahan tersebut. Tampilkan dan jelaskan secara lengkap langkah-langkah yang anda lakukan, dimulai dari membuka Python mode interaktif di komputer anda masing-masing.

Latihan 1.3 Berkaitan dengan compound interest pada Contoh 1.2 dan 1.3, jika Erika memiliki uang 200 juta rupiah dan ingin disimpan di deposito Pasti Cuan sampai uangnya menjadi **minimal** 400 juta, berapa lama waktu yang dibutuhkan?

catatan: bunga 10% per-tahun.

Gunakan Python mode interaktif untuk menyelesaikan permasalahan tersebut. Tampilkan dan jelaskan secara lengkap langkah-langkah yang anda lakukan, dimulai dari membuka Python mode interaktif di komputer anda masing-masing.



2. Variable, Expression dan Statements

2.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan dan menggunakan variabel, ekspresi, dan statement pada Python.

2.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Editor Visual Studio Code, Repl Python 3 (online), pyCharm, atau Spyder

2.3 Materi

2.3.1 Values dan type

Value merupakan komponen utama dari program, seperti huruf atau angka. *Value* yang sering kita kenal misalnya 1,2,'a','z', dan "Hello Word". *Value* dibagi menjadi beberapa tipe yang berbeda, misalnya 2 untuk sebuah nilai integer dan "Hello Word" untuk sebuah nilai *string*. Interpreter dapat melakukan identifikasi terhadap *string* karena pada penulisannya ditutup menggunakan tanda petik (*quotation mark*).

Untuk mencoba memahami values dan type, silakan coba beberapa baris kode berikut ini menggunakan python interactive mode seperti yang ditunjukkan pada potongan kode dibawah ini.

```
>>> print(4)
4
>>> print(10.876)
10.876
```

```
>>> print('Z')
Z
>>> print('True')
True
>>> print('False')
False
```

Perintah print juga bekerja untuk value selain *string*, seperti *integer* (bilangan bulat), *float* (bilangan pecahan), *character* (huruf), atau *bool* (benar/salah). Untuk mencobanya kita dapat menggunakan perintah python untuk menjalankan interpreter.

Setiap value yang pasti memiliki type untuk mengetahui tipe data tersebut. Python menyediakan fungsi built-in untuk melakukan pengecekan tipe data pada value dengan menggunakan fungsi *type()*

```
>>> x=5
>>> print(x, "tipenya adalah ", type(x))
5 tipenya adalah <class int'>
>>> x = 2.0
>>> print(x, "tipenya adalah ", type(x))
2.0 tipenya adalah <class float'>
>>> x= 1+2j
>>> print(x, "tipenya adalah ",type(x))
(1+2j) tipenya adalah <class complex">
```

Ketika menggunakan bilangan bulat besar, beberapa model penulisan menggunakan tanda koma (,) diantara kelompok tiga digit. Misalnya pada penulisan 1.000.000. Dalam python, akan dianggap sebagai bilangan bulat.

```
>>> print(1,000,000)
1,0,0
```

Hal ini terjadi karena Python menganggap bahwa 1,000,000,000 merupakan kiriman parameter sebanyak 3 parameter pada fungsi print, yaitu 1, 0, dan 0.

2.3.2 Variabel

Salah satu fitur powerfull dalam bahasa pemrograman adalah kemampuannya untuk melakukan manipulasi *variable*. *Variable* merupakan lokasi memori yang dicadangkan untuk menyimpan nilai-nilai. Ini berarti bahwa ketika Anda membuat sebuah *variable* Anda memesan beberapa ruang di memori. *Variable* menyimpan data yang dilakukan selama program dieksekusi, yang nantinya isi dari variabel tersebut dapat diubah oleh operasi - operasi tertentu pada program yang menggunakan *variable*.

```
>>> pesan = 'selamat pagi, mari belajar python'
>>> n = 17
>>> pi = 3.1415926535897931
```

Variable dapat menyimpan berbagai macam tipe data. Di dalam pemrograman Python, *variable* mempunyai sifat yang dinamis, artinya *variable* Python tidak perlu dideklarasikan tipe data tertentu dan *variable* Python dapat diubah saat program dijalankan.

Potongan kode diatas merupakan contoh penggunaan dari *varibale*. Contoh pertama adalah *variable* pesan yang berisi string, contoh kedua adalah *variale* n yang berisi nilai integer 17 dan contoh ketiga merupakan nilai dari pi (π). Untuk menampilkan nilai dari *varibale*, dapat digunakan perintah print.

```
>>> print(n)
17
>>> print(ipk)
3.29
```

2.3.3 Nama Variabel dan Keywords

Pemberian nama pada variabel mengacu pada panduan berikut ini.

1. Nama variable boleh diawali menggunakan huruf atau garis bawah (_), contoh: nama, _nama, namaKu, nama_variable.
2. Karakter selanjutnya dapat berupa huruf, garis bawah (_) atau angka, contoh: _nama, n2, nilai1.
3. Karakter pada nama variable bersifat sensitif (*case-sensitif*). Artinya huruf besar dan kecil dibedakan. Misalnya, variabel_Ku dan variabel_ku, keduanya adalah variabel yang berbeda.
4. Nama variabel tidak boleh menggunakan kata kunci yang sudah ada dalam python seperti if, while, for, dsb.

Python sendiri memiliki 35 *keyword* yang tidak boleh digunakan untuk memberi nama variabel.

and	del	from	None	True
as	elif	global	nonlocaly	try
assert	else	if	not	while
break	except	import	or	width
class	False	in	pass	yield
continue	finally	is	raise	async
def	for	lamda	return	wait

Berikut ini contoh penggunaan variable dalam bahasa pemrograman Python.

```
#proses memasukan data ke dalam variabel
nama = "Agung Sejagat"

#proses mencetak variabel
print(nama)

#nilai dan tipe data dalam variabel dapat diubah
umur = 20
print(umur)
type(umur)
umur = "dua puluh satu" #nilai setelah diubah
print(umur) #mencetak nilai umur
type(umur) #mengecek tipe data umur
namaDepan = "Joko"
namaBelakang = "Widodo"
nama = namaDepan + " " + namaBelakang
umur = 22
hobi = "Berenang"
print("Biodata\n", nama, "\n", umur, "\n", hobi)

#contoh variabel lainnya
inivariabel = "Halo"
```

```

ini_juga_variabel = "Hai"
_inivariabeljuga = "Hi"
inivariabel222 = "Bye"
panjang = 10
lebar = 5
luas = panjang * lebar
print(luas)

```

2.3.4 Statements

Statements merupakan bagian dari code interpreter Python yang dapat dieksekusi. Misalnya pada statement print, dapat berupa *expression statements* dan *assignment*.

Ketika menggunakan python dalam mode interaktif, interpreter secara langsung akan melakukan eksekusi dan menampilkan hasilnya. Hal ini tentu saja berbeda ketika menggunakan *script mode*. *Script* biasanya berisi *statements* yang saling berhubungan secara sekuensial.

```

1 print(1)
2 x=2
3 print(x)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

192:~ macintosh$ /Library/Frameworks/Python.framework/Versions/3.7/bin/python3 "
python/source-code/bab 02/statemnet.py"
1
2
192:~ macintosh$ █

```

Gambar 2.1: Contoh statement dan outputnya

2.3.5 Operator dan Operand

Operator adalah simbol tertentu yang digunakan untuk melakukan operasi aritmatika maupun logika. Nilai yang padanya dilakukan operasi disebut *operand*. Misalnya adalah $2 + 3$. Di sini tanda $+$ adalah operator penjumlahan. 2 dan 3 adalah operand.

Pada bagian ini secara khusus akan membahas operator aritmatika pada Python. Operator aritmatika adalah operator yang digunakan untuk melakukan operasi matematika, seperti penjumlahan, pengurangan, perkalian, pembagian, dan sebagainya. Tabel berikut menunjukkan jenis operator aritmatika.

Tabel 2.1: Operator pada Python

Operator	Nama dan Fungsi	Contoh
+	Penjumlahan, menjumlahkan 2 buah operand	$x + y$
-	Pengurangan, mengurangi 2 buah operand	$x - y$
*	Perkalian, mengalikan 2 buah operand	$x * y$
/	Pembagian, membagi 2 buah operand	x / y
**	Pemangkatan, memangkatkan bilangan	$x ** y$

Beberapa contoh penggunaan operator aritmatika pada Python dapat dilihat pada potongan kode berikut ini

```

>>>32 + 30
62

```



```
>>> hour = 5
>>> print (hour-1)
4
>>> minute=60
>>> print (hour*6+minute)
90
>>> print (minute/60)
1.0
>>> 5**2
25
>>> (5+9)*(15-7)
112
```

2.3.6 Expressions

Expression merupakan representasi dari nilai dan dapat terdiri dari gabungan antara *values*, *variable* dan *operator*. *Values* dengan sendirinya dapat dianggap sebagai *expression* dan juga variabel. Secara umum, semuanya dapat disebut dengan *expression*.

```
17
x
x+17
```

Ketika menggunakan *expression* dalam model interactive, interpreter akan melakukan evaluasi dan menampilkan hasilnya.

```
>>> 1 + 1
2
>>> 3 + 2
5
```

2.3.7 Urutan Operasi

Urutan operasi berlaku bila ada lebih dari satu operator dalam *expression*. Urutan operasi bergantung pada aturan prioritas. Untuk operasi matematika, Python mengikuti konversi matematika. Urutan operasi sering disingkat dengan **PEMDAS - Parentheses, Exponentiation, Multiplication and Division, Operator**.

- *Paranthese* (Tanda kurung) - merupakan prioritas tertinggi dan digunakan untuk memaksa *expression* dalam urutan yang sesuai. Contohnya $2*(3-1)$ hasilnya 4, dan $(1+1) ** (5-2)$ hasilnya 8. Penggunaan tanda kurung dapat digunakan untuk membuat *expression* menjadi lebih mudah untuk dibaca, misal $(minute * 100) / 60$.
- *Exponentiation* (Eksponensial/Pemangkatan) - merupakan prioritas tertinggi berikutnya, contoh $2**1+1$ hasilnya 3, bukan 4, dan $3*1**3$ hasilnya 3 bukan 27
- *Multiplication and Divison* (Perkalian dan Pembagian) - memiliki prioritas yang sama tetapi lebih tinggi dari penjumlahan dan pengurangan. Penjumlahan dan pengurangan juga memiliki prioritas yang sama pula. Contoh $2*3-1$ hasilnya 5 bukan 4, dan $6+4/2$ hasilnya 8, bukan 5.
- *Operators* - operator memiliki prioritas yang sama, dibaca dari kiri ke kanan. Contoh $5-3-1$ hasilnya 1 bukan 3 karena operasi pengurangan $5-3$ terlebih dahulu baru kemudian hasilnya dikurangi dengan 1.

Jika terjadi keraguan, silakan letakkan tanda kurung di dalam ekspresi untuk memastikan bahwa komposisinya sesuai dengan yang diinginkan.

2.3.8 Operator Modulus dan String

Modulus

Operator Modulus merupakan sisa hasil bagi dari bilangan pertama dengan bilangan kedua. Operator ini hanya berlaku pada tipe data integer. Dalam python, operator modulus dilambangkan dengan tanda persen (%).

```
>>> quotient = 7 // 3
>>> print(quotient)
2
>>> oprmomulus = 7 % 3
>>> print(oprmomulus)
1
```

7 dibagi dengan 3 menghasilkan 2 dnegan sisa hasil bagi 1.

Contoh penggunaan operator modulus.

- Memeriksa satu angka dapat dibagi dengan yang lain, misal jika $x \% y$ adalah 0, maka x dapat dibagi oleh y .
- Dapat mengekstrak digit paling kanan atau digit dari suatu angka. Misalnya, $x \% 10$ menghasilkan digit x paling kanan (dalam basis 10). Demikian pula, $x \% 100$ menghasilkan dua digit terakhir.

String

Operator + ketika bekerja dengan string tidak berarti penjumlahan secara matematika, melainkan penggabungan antar string. Contoh:

```
>>> first = 10
>>> second = 15
>>> print(first+second)
25
>>> first = '100'
>>> second = '150'
>>> print(first + second)
100150
```

Operator * juga bekerja dengan string dengan melakukan perkalian antara content string dan integer.

```
>>> first = 'Test '
>>> second = 3
>>> print(first * second)
Test Test Test
```

2.3.9 Menangani Input dari Pengguna

Sebuah program biasanya memiliki alur kerja Input - Proses - Output yang alurnya ditunjukkan pada Gambar 2.2.

Input adalah data/masukan yang dibutuhkan supaya program bisa berjalan. Proses adalah langkah-langkah yang dilakukan oleh program untuk memecahkan masalah. Sedangkan Output adalah hasil yang didapatkan setelah menjalankan langkah-langkah tersebut. Sebagai contoh misalnya mengambil uang lewat ATM. Pengambilan uang melalui ATM dapat dibagi menjadi 3 bagian (Input-Proses-Output) tersebut, yaitu:



Gambar 2.2: Input - Proses - Output.

1. Masukkan kartu ATM. Masukkan PIN anda. Kartu ATM dan PIN adalah Input yang diperlukan supaya anda bisa mengambil uang.
2. Anda memilih menu Pengambilan Uang. Kemudian anda memasukkan nominal yang diinginkan. Bagian ini juga merupakan Input.
3. Mesin ATM akan memproses transaksi anda dengan menghubungi server Bank yang bersangkutan. Dilakukan berbagai macam pengecekan (misal: apakah saldonya cukup? apakah kartunya masih berlaku? apakah ada blokir?). Bagian ini disebut Proses.
4. Mesin ATM mengeluarkan uang, bukti pengambilan uang dan kartu anda. Uang yang keluar merupakan Output dari kegiatan ini. Selain itu, saldo anda juga berkurang. Pengurangan saldo juga merupakan hasil dari kegiatan ini.

Python juga dapat menangani input dari pengguna. Input dalam hal ini dapat berupa input text yang dimasukkan oleh pengguna. Untuk itu python menyediakan built-in function yang disebut `input` untuk mendapatkan input dari keyboard. Ketika fungsi ini dipanggil, program akan berhenti dan menunggu pengguna untuk mengetik sesuatu. Ketika pengguna menekan tombol Enter, program akan dilanjutkan dan input akan mengembalikan apa yang diketik oleh pengguna sebagai string.

```
>>> inp = input()
Pada hari minggu kuturut ayah ke kota
>>> print(inp)
Pada hari minggu kuturut ayah ke kota
```

Sebelum mendapatkan input dari pengguna, lebih baik untuk mencetak prompt yang memberitahu pengguna apa yang harus diinput. String tersebut dapat diteruskan ke input untuk ditampilkan kepada pengguna sebelum berhenti untuk input.

```
>>> name = input('Siapa nama mu ?\n')
Siapa nama mu ?
Sancaka
>>> print(name)
Sancaka
```

Tanda `\n` pada akhir prompt mewakili baris baru atau ganti baris sehingga input pengguna muncul dibawah prompt.

Ketika mengharapkan pengguna untuk mengetik bilangan bulat, dapat dilakukan dengan mengonversi nilai kembali ke int menggunakan fungsi `int()`:

```
>>> prompt = 'Berapa suhu ruangan sekarang?\n'
>>> suhu = input(prompt)
Berapa suhu ruangan sekarang?
24
>>> int(suhu)
```

```
24
>>> int(suhu) + 5
29
```

Akan terjadi error jika pengguna memasukkan data selain angka.

```
>>> prompt = 'Berapa suhu ruangan sekarang?\n'
>>> suhu = input(prompt)
Berapa suhu ruangan sekarang?
Ruangan depan atau belakang ?
>>> int(suhu)
ValueError: invalid literal for int() with base 10:
```

2.4 Komentar

Tanda pagar (#) digunakan untuk menandai komentar di python. Komentar tidak akan diproses oleh interpreter Python. Komentar hanya berguna untuk programmer untuk memudahkan memahami maksud dari kode.

```
# Komentar pertama
print("hai dunia!") # Komentar kedua
```

Outputnya

```
hai dunia
```

Python tidak memiliki fitur komentar multibaris. Kita harus mengomentari satu persatu baris seperti berikut:

```
# Ini komentar
# Ini juga adalah komentar
# Ini juga masih komentar
```

2.5 Kegiatan Praktikum

Kegiatan praktikum yang akan dilakukan adalah sebagai berikut:

1. Membuat variabel.
2. Memberikan nilai dalam variabel.
3. Mencetak nilai dalam variabel.
4. Separator, tipe data dan fungsi type

2.5.1 Membuat Variabel

Misalkan sebuah data pribadi berisi nama, alamat, umur, tempat lahir, tanggal lahir, indeks prestasi kumulatif akan memberikan 6 (enam) buah variabel dengan tipe datanya.

```
1 Nama = input("Nama Anda : ")
2 Alamat = input("Alamat Tinggal Anda : " )
3 Umur = input("Umur Anda : ")
4 TL = input("Tempat Lahir Anda : ")
5 Tgl = input("Tanggal Lahir Anda : ")
```

```

6 IPK = input("IPK Anda : ")
7 print("=====")
8 print("DATA AKADEMIK")
9 print("Nama Anda : ", Nama)
10 print("Alamat Tinggal Anda : ",Alamat)
11 print("Umur Anda : ", Umur)
12 print("Tempat Lahir Anda : ", TL)
13 print("Tanggal Lahir Anda : ", Tgl)
14 print("IPK Anda :", IPK)

```

Output dari program diatas dapat dilihat pada gambar 2.3

```

192:~ macintosh$ /Library/Frameworks/Python.framework/Versions/3.7/bin/python3 "/Users/macintosh/Documents/GitHub/modul-a
ode/bab 02/lat_input_output.py"
Nama Anda : Wahyu Sardono
Alamat Tinggal Anda : Klaten
Umur Anda : 17
Tempat Lahir Anda : Klaten
Tanggal Lahir Anda : 19 Agustus 1986
IPK Anda : 9
=====
DATA AKADEMIK
Nama Anda : Wahyu Sardono
Alamat Tinggal Anda : Klaten
Umur Anda : 17
Tempat Lahir Anda : Klaten
Tanggal Lahir Anda : 19 Agustus 1986
IPK Anda : 9
192:~ macintosh$ 

```

Gambar 2.3: Contoh input/output tipe data string.

Bagaimana dengan tipe data yang lain? Kita akan mencoba untuk membuat program dengan menggunakan tipe data bilangan baik integer maupun float.

```

1  # CONTOH PROGRAM
2  # MENGHITUNG BILANGAN
3  #=====
4
5  x1 = eval(input("X1 = "))
6  x2 = eval(input("X2 = "))
7  x3 = eval(input("X3 = "))
8  x4 = eval(input("X4 = "))
9
10 jumlah = x1+x2+x3+x4
11 kali = x1*x2*x3*x4
12 print('Hasil Penjumlahan semua bilangan = ', jumlah)
13 print('Hasil Perkalian semua bilangan = ', kali)
14 jumlah = jumlah + 0.5
15 print('Jika ditambah 0.5 hasilnya = ',jumlah)
16 kali = kali * 0.5
17 print('Jika dikali 0.5 hasilnya = ',kali)

```

Output dari program diatas dapat dilihat pada gambar 2.4

```

didanendya:~ macintosh$ /Library/Frameworks/Python.framework/Versions/3.7/bin/python3 "/Us
source-code/bab 02/hitung_bil.py"
X1 = 2
X2 = 5
X3 = 7
X4 = 9
Hasil Penjumlahan semua bilangan = 23
Hasil Perkalian semua bilangan = 630
Jika ditambah 0.5 hasilnya = 23.5
Jika dikali 0.5 hasilnya = 315.0

```

Gambar 2.4: Contoh input/output tipe data bilangan.

Dari gambar 2.4 dapat diketahui bahwa input/output pada tipe data bilangan dengan hasil yang berbeda tipe bilangannya yaitu tipe integer (bilangan bulat) atau float (bilangan berkoma).

`eval()` digunakan untuk melakukan konversi expression dari **string** menjadi **integer**.

2.5.2 Memberikan nilai dalam variabel

Lakukan inisiasi variabel atau konstanta dari permasalahan berikut! Menjumlahkan total harga pada saat konsumen membeli beberapa barang.

Langkah 1 : Inisiasi Persoalan

Variabel/konstanta input :

kode_barang, nama_barang, harga_satuan_barang,

jumlah_per_barang_beli, total_harga_per_transaksi = 0 Proses :

harga_beli_per_barang = harga_satuan_barang * jumlah_per_barang_beli

total_harga_per_transaksi=harga_beli_per_barang + total_harga_per_transaksi

Output :

total_harga_per_transaksi

Langkah 2 : Menetapkan Tipe Data

kd_brg, nama_brg bertipe data *string*

jum_brg bertipe data *integer*

harga_satuan, harga_beli, total_hrg_brg bertipe data *float*.

Langkah 3 : Kode Program

```

1 total_hrg_brg= 0.0
2 kd_brg=input("Kode barang = ")
3 nama_brg=input("Nama barang = ")
4 harga_satuan=eval(input("Harga satuan barang =Rp. "))
5 jum_brg=eval(input("Jumlah barang yang dibeli = "))
6 harga_beli = harga_satuan * jum_brg
7 total_hrg_brg= harga_beli + 123total_hrg_brg
8 print("Total harga yang dibayar Rp",total_hrg_brg)

```

Output dari program diatas dapat dilihat pada gambar 2.5

```

192:bab 02 macintosh$ python3 lat_variabel.py
Kode barang = 445567
Nama barang = Gunting Kuku
Harga satuan barang =Rp. 1750
Jumlah barang yang dibeli = 20
Total harga yang dinayar Rp 35000.0
192:bab 02 macintosh$ █

```

Gambar 2.5: Contoh input/output menggunakan variabel.

2.5.3 Mencetak nilai dalam variabel

Mencetak nilai dalam sebuah variabel menggunakan perintah print.

```

1  x=30
2  print(x)
3  type(x)
4
5  y = 4*int(x)
6  print(y)
7
8  print("Tipe data X dikonversi ke int agar dapat dihitung Y=4*x")
9
10 z = y+float(x)
11 print("z",z)
12
13 print("Tipe data X dikonversi menjadi float untuk dijumlahkan
14 dengan tipe data Y, hasilnya yang ditampung
15 bertipe float juga")
16
17 P=True
18 L=False
19
20 P!=L
21 print("Tanda != artinya tidak sama dengan")
22
23 P==L
24 print("Tanda == artinya sama dengan")

```

2.5.4 Separator, tipe data dan fungsi type

Konversi type data pada pemrograman python gunakan fungsi berikut :

1. **str()** = Untuk konversi type data ke String
2. **int()** = Untuk konversi type data ke Integer
3. **float()** = Untuk konversi type data ke Float

Ada dua macam variasi print :

1. Jika ada simbol, gunakan kutip dua atau gunakan backslash (\) sebelum menuliskan simbol.
2. Dipisahkan dengan tanda koma.

3. Diganti dengan :

- %d : mewakili integer
- %f : mewakili float
 - Untuk membuat n angka di belakang koma, gunakan %.nf
 - Misal untuk dua angka di belakang koma, berarti gunakan %.2f
- %s : mewakili string

```
1 Kode='AB123'
2 NamaBarang='Kaca Mata'
3 HargaSatuan=125000
4 Stok=10
5 print('Kode barang %s \n Nama Barang= %s \n Harga Satuan=Rp %d \n Stok Barang=%d' %
6 (Kode,NamaBarang,HargaSatuan,Stok))
7
8 HargaBarang=float(HargaSatuan)
9 print('Harga Satuan Barang= Rp. %.3f' %(HargaBarang))
```

Hasil dari potongan kode program diatas dapat dilihat pada gambar 2.6

```
didanendya:~ macintosh$ /Library/Frameworks/Python.framework/Versions/3.7/bin/python3 "/Users/macintosh/Documents/GitHub/
source-code/bab_02/coba_tipe_data.py"
Kode barang AB123
Nama Barang= Kaca Mata
Harga Satuan=Rp 125000
Stok Barang=10
Harga Satuan Barang= Rp. 125000.000
didanendya:~ macintosh$
```

Gambar 2.6: Tampilan Contoh print Tipe Data String, Integer dan Float.

2.6 Latihan Mandiri

Latihan 2.1 Buatlah program yang dapat menghitung berat badan yang diperlukan, jika diketahui tinggi badan dan nilai Body Mass Index (BMI) yang diharapkan! Body Mass Index dihitung dengan cara: $BMI = \frac{\text{berat}}{\text{tinggi}^2}$. Perhatikan, berat badan dalam satuan kilogram (kg) dan tinggi badan dalam satuan meter (m). ■

Latihan 2.2 Buatlah program yang dapat menghitung hasil dari fungsi $f(x) = 2x^3 + 2x + \frac{15}{x}$, di mana x merupakan bilangan bulat yang dimasukkan oleh pengguna. ■

Latihan 2.3 Budi tertarik untuk melamar pekerjaan pada liburan semester yang akan berlangsung selama 5 minggu. Gaji yang diberikan adalah gaji per jam. Total pajak yang harus budi bayarkan dari penghasilannya selama bekerja adalah 14%. Setelah membayar pajak, budi menghabiskan 10% dari pendapatan bersihnya untuk membeli baju dan aksesoris yang akan digunakan pada semester baru, dan 1% untuk membeli alat tulis. Setelah membeli baju, aksesoris dan alat tulis, Budi menggunakan 25% dari sisa uangnya untuk disedekahkan. Setiap Rp.1000 yang Budi sedekahkan 30% nya akan diserahkan kepada anak yatim, dan sisanya akan diserahkan ke kaum dhuafa.

Buatlah sebuah program, dengan input:

1. Gaji per jam yang anda inginkan
2. Jumlah jam kerja yang akan dilakukan dalam 1 minggu

Output dari program adalah sebagai berikut :

1. Pendapatan Budi selama libur musim panas sebelum melakukan pembayaran pajak.
2. Pendapatan Budi selama libur musim panas setelah melakukan pembayaran pajak.
3. Jumlah uang yang akan Budi habiskan untuk membeli pakaian dan aksesoris.
4. Jumlah uang yang akan Budi habiskan untuk membeli alat tulis.
5. Jumlah uang yang akan Budi sedekahkan.
6. Jumlah uang yang akan diterima anak yatim.
7. Jumlah uang yang akan diterima kaum dhuafa.



Struktur Kontrol dan Modular Programming

3	Struktur Kontrol Percabangan	35
3.1	Tujuan Praktikum	
3.2	Alat dan Bahan	
3.3	Materi	
3.3.1	Boolean Expression dan Logical Operator	
3.3.2	Bentuk-bentuk Percabangan	
3.3.3	Penanganan Kesalahan Input Menggunakan Exception Handling	
3.4	Kegiatan Praktikum	
3.4.1	Contoh Masalah-masalah Percabangan	
3.5	Latihan Mandiri	
4	Modular Programming	47
4.1	Tujuan Praktikum	
4.2	Alat dan Bahan	
4.3	Materi	
4.3.1	Fungsi, Argument dan Parameter	
4.3.2	Return Value	
4.3.3	Optional Argument dan Named Argument	
4.3.4	Anonymous Function (Lambda)	
4.4	Kegiatan Praktikum	
4.4.1	Mendefinisikan Fungsi	
4.4.2	Argument dan Parameter	
4.4.3	Anonymous/Lambda Function	
4.5	Latihan Mandiri	
5	Struktur Kontrol Perulangan	57
5.1	Tujuan Praktikum	
5.2	Alat dan Bahan	
5.3	Materi	
5.3.1	Definisi Perulangan	
5.3.2	Bentuk Perulangan for	
5.3.3	Bentuk Perulangan While	
5.3.4	Penggunaan Break dan Continue	
5.3.5	Konversi dari Bentuk for Menjadi Bentuk while	
5.4	Kegiatan Praktikum	
5.4.1	Deret Bilangan	
5.4.2	Penggunaan Break	
5.5	Latihan Mandiri	
6	Percabangan dan Perulangan Kompleks	67
6.1	Tujuan Praktikum	
6.2	Alat dan Bahan	
6.3	Materi	
6.3.1	Struktur Percabangan Kompleks	
6.3.2	Struktur Perulangan Kompleks	
6.4	Kegiatan Praktikum	
6.5	Latihan Mandiri	

3. Struktur Kontrol Percabangan

3.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menyusun boolean expression dengan benar.
2. Dapat merangkai beberapa boolean expression menggunakan operator logical dengan benar.
3. Dapat menyusun percabangan dengan if sesuai dengan permasalahan yang dihadapi.
4. Dapat menangani jika terjadi kesalahan masukan pengguna menggunakan bentuk exception sederhana.

3.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

3.3 Materi

3.3.1 Boolean Expression dan Logical Operator

Perhatikan kasus berikut: Voucher diskon 30% dapat dipakai jika minimum pembelian anda adalah Rp. 100.000. Minimum pembelian adalah syarat yang harus dipenuhi untuk mendapatkan diskon. Minimum pembelian tersebut dapat dinyatakan dalam Python sebagai berikut:

```
pembelian >= 100000
```

Bentuk tersebut dinamakan sebagai boolean expression, karena hasilnya hanya ada dua kemungkinan, yaitu **True** atau **False**, tergantung dari variabel pembelian. Jika anda memasukkan perintah tersebut ke dalam Python mode interaktif, hasilnya dapat dilihat pada Gambar 3.1.

```
yuanlukito@linuxstation: ~  
yuanlukito@linuxstation:~$ python3  
Python 3.7.5 (default, Nov 20 2019, 09:21:52)  
[GCC 9.2.1 20191008] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> pembelian = 120000  
>>> pembelian >= 100000  
True  
>>>  
>>> pembelian = 45000  
>>> pembelian >= 100000  
False  
>>>  
>>> pembelian = 100000  
>>> pembelian >= 100000  
True  
>>> █
```

Gambar 3.1: Kemungkinan hasil dari boolean expression.

Boolean expression dapat disusun menggunakan operator-operator perbandingan seperti pada Tabel 3.1.

Tabel 3.1: Operator-operator perbandingan (comparison).

Operator	Keterangan
x == y	Apakah x sama dengan y?
x != y	Apakah x tidak sama dengan y?
x > y	Apakah x lebih besar dari y?
x >= y	Apakah x lebih besar atau sama dengan y?
x < y	Apakah x lebih kecil dari y?
x <= y	Apakah x lebih kecil atau sama dengan y?
x is y	Apakah x sama dengan y?
x is not y	Apakah x tidak sama dengan y?

Anda harus bisa menyusun bentuk boolean expression dan memilih operator yang tepat sesuai dengan permasalahan yang dihadapi. Beberapa hal yang perlu anda perhatikan saat menyusun bentuk boolean expression:

- Bentuk boolean expression pasti hasilnya **hanya ada dua**, yaitu True atau False.
- Perhatikan kata-kata khusus seperti **minimum, maksimum, tidak lebih dari, tidak kurang dari, tidak sama, tidak berbeda**.
- Perhatikan dengan seksama dan tentukan variabel yang perlu dibandingkan dengan benar sesuai dengan permasalahan.

Beberapa contoh permasalahan dan bentuk boolean expression-nya dapat dilihat pada Tabel 3.2.

Beberapa boolean expression dapat digabungkan dengan menggunakan logical operator. Logical operator pada Python adalah **and**, **or** dan **not**. Sebagai contoh, misalnya wahana Rollercoaster

Tabel 3.2: Operator-operator perbandingan (comparison).

Contoh masalah	Boolean expression
Untuk lulus dibutuhkan IPK minimum 2.25	<code>ipk >= 2.25</code>
Golden Button hanya diberikan untuk Youtuber dengan subscriber lebih dari 1 juta	<code>subscriber > 1000000</code>
Pengendara dengan kecepatan lebih dari 90 km/jam akan mendapatkan tilang	<code>kecepatan > 90</code>
Wahana Rollercoaster hanya bisa dinaiki oleh mereka yang tinggi badannya lebih dari 110 cm	<code>tinggi > 110</code>
Nilai ujian Hanna adalah 75 sedangkan Robby mendapatkan nilai 75. Apakah nilai keduanya sama?	<code>hanna is robbby</code>
Junaedi memiliki 10 sepatu, Ricky punya 15 sepatu dan Arnold punya 20 sepatu. Apakah gabungan sepatu Junaedi dan Ricky lebih banyak dari sepatu milik Arnold?	<code>junaedi + ricky > arnold</code>

hanya dapat dinaiki oleh penumpang dengan usia minimal 10 tahun dan tinggi badan minimal 110 cm. Kedua persyaratan tersebut dapat dinyatakan dalam bentuk sebagai berikut:

```
usia >= 10 and tinggi >= 110
```

Diskon diberikan kepada member atau jumlah pembelian lebih dari Rp. 500.000:

```
member == true or pembelian > 500000
```

3.3.2 Bentuk-bentuk Percabangan

Percabangan pada Python secara umum ada tiga bentuk, yaitu: **conditional**, **alternative** dan **chained conditional**. Bentuk conditional secara umum dinyatakan dalam kode program sebagai berikut:

```
if <kondisi>:
    <lakukan ini>
    <lakukan ini>
    ...
```

Sebagai contoh, jika nilai akhir > 70 maka akan mendapatkan sertifikat kelulusan. Kode programnya sebagai berikut:

```
if nilai_akhir > 70:
    print("Anda lulus dan mendapatkan sertifikat kelulusan!")
```

Bentuk alternative conditional adalah bentuk percabangan yang memiliki dua alternatif langkah yang harus dijalankan berdasarkan kondisi tertentu. Secara umum bentuknya adalah sebagai berikut:

```
if <kondisi>:
    <lakukan ini>
    <lakukan ini>
    ...
else:
    <lakukan itu>
    <lakukan itu>
    ...
```

Sebagai contoh, jika nilai akhir > 60, tampilkan tulisan Lulus. Jika tidak, tampilkan tulisan Tidak Lulus. Pada contoh ini, ada dua kemungkinan tulisan yang muncul, yaitu Lulus atau Tidak Lulus. Implementasinya sebagai berikut:

```
if nilai_akhir > 60:
    print("Lulus")
else:
    print("Tidak Lulus")
```

Bentuk chained conditional digunakan jika kemungkinan langkah yang harus dijalankan berikutnya lebih dari dua. Bentuknya secara umum adalah sebagai berikut:

```
if <kondisi 1>:
    <lakukan A1>
    <lakukan A2>
    ...
elif <kondisi 2>:
    <lakukan B1>
    <lakukan B2>
    ...
elif <kondisi 3>:
    <lakukan C1>
    <lakukan C2>
    ...
...
else
    <lakukan ...>
    <lakukan ...>
    ...
```

Sebagai contoh, misalnya sebuah toko pakaian memberi diskon yang besarnya ditentukan oleh nilai pembelian anda. Untuk pembelian di atas Rp. 1.000.000 mendapatkan diskon 30%. Pembelian lebih dari Rp. 500.000 sampai Rp. 1.000.000 mendapatkan diskon 20%. Pembelian dari Rp. 100.000 sampai Rp. 500.000 mendapatkan diskon 15%. Pembelian di bawah Rp. 100.000 tidak mendapatkan diskon. Kemungkinan diskon yang diberikan ada 4, yaitu: 30%, 20%, 15% dan 0% (tidak diskon). Bagaimana implementasinya? Perhatikan implementasi berikut ini:

```
if pembelian > 1000000:
    diskon = 0.3          # diskon 30%
elif pembelian > 500000 and pembelian <= 1000000:
    diskon = 0.2          # diskon 20%
elif pembelian >= 100000 and pembelian <= 500000:
    diskon = 0.15         # diskon 15%
else:
    diskon = 0            # tidak ada diskon
```

Pernyataan "Pembelian lebih dari Rp. 500.000 sampai Rp. 1.000.000 mendapatkan diskon 20%" diimplementasikan dalam bentuk gabungan dari dua boolean expression, yaitu:

```
pembelian > 500000 and pembelian <= 1000000
```


Umumnya untuk menyatakan rentang nilai tersebut kita menggunakan logical operator and, karena kedua kondisi tersebut harus terpenuhi agar bernilai True.

Selain bentuk-bentuk percabangan tersebut, Python juga memiliki sintaks alternatif untuk menuliskan percabangan yang biasa disebut sebagai **ternary operator**. Sebagai contoh bentuk percabangan berikut ini:

```
pembelian = int(input("Jumlah pembelian: "))
diskon = 0.1 if pembelian > 100000 else 0
```

Bentuk ternary tersebut merupakan bentuk lain dari if-else berikut ini:

```
pembelian = int(input("Jumlah pembelian: "))
if pembelian > 100000:
    diskon = 0.1
else:
    diskon = 0
```

3.3.3 Penanganan Kesalahan Input Menggunakan Exception Handling

Dalam menangani input dari pengguna, kita juga perlu memperhatikan potensi kesalahan yang mungkin terjadi sehingga program tidak bisa berjalan dengan semestinya. Untuk lebih jelasnya, perhatikan program yang meminta input usia pengguna, kemudian program akan menampilkan apakah pengguna termasuk lansia, dewasa, remaja, kanak-kanak atau balita. Kategori usia tersebut mengikuti aturan sebagai berikut:

- Balita: 0-5 tahun.
- Kanak-kanak: 6-11 tahun.
- Remaja: 12-25 tahun.
- Dewasa: 26-45 tahun.
- Lansia: > 45 tahun.

Program tersebut adalah sebagai berikut:

```
1  usia = int(input("Masukkan usia anda: "))
2  if usia <= 5:
3      print("Balita")
4  elif usia >= 6 and usia <= 11:
5      print("Kanak-kanak")
6  elif usia >=12 and usia <= 25:
7      print("Remaja")
8  elif usia >= 26 and usia <= 45:
9      print("Dewasa")
10 elif usia > 45:
11     print("Lansia")
```

Jika program tersebut dijalankan beberapa kali dengan input yang berbeda-beda, hasilnya telah sesuai dengan yang diharapkan. Hasil dari program tersebut dapat dilihat pada Gambar 3.2.

Bagian dari program tersebut yang meminta pengguna memasukkan usianya ada di baris 1, yaitu:

```
usia = int(input("Masukkan usia anda: "))
```

```
yuanlukito@linuxstation: ~/Documents
yuanlukito@linuxstation:~/Documents$ python3 kategoriusia.py
Masukkan usia anda: 33
Dewasa
yuanlukito@linuxstation:~/Documents$ python3 kategoriusia.py
Masukkan usia anda: 7
Kanak-kanak
yuanlukito@linuxstation:~/Documents$ python3 kategoriusia.py
Masukkan usia anda: 68
Lansia
yuanlukito@linuxstation:~/Documents$ python3 kategoriusia.py
Masukkan usia anda: 18
Remaja
yuanlukito@linuxstation:~/Documents$ python3 kategoriusia.py
Masukkan usia anda: 2
Balita
yuanlukito@linuxstation:~/Documents$ █
```

Gambar 3.2: Hasil program kategori usia.

Seperti yang telah dipelajari pada Bab sebelumnya, fungsi **input()** digunakan untuk membaca masukan/input yang diberikan oleh pengguna. Fungsi **input()** akan menghasilkan string, sedangkan usia seharusnya merupakan angka/bilangan sehingga perlu dikonversi menjadi bilangan bulat dengan fungsi **int()**. Program tersebut akan berjalan dengan baik selama pengguna tidak memasukkan input yang salah atau tidak sesuai yang diharapkan. Sebagai contoh, perhatikan Gambar 3.3 yang menunjukkan pengguna memasukkan input yang tidak sesuai.

```
yuanlukito@linuxstation: ~/Documents
yuanlukito@linuxstation:~/Documents$ python3 kategoriusia.py
Masukkan usia anda: dua puluh
Traceback (most recent call last):
  File "kategoriusia.py", line 1, in <module>
    usia = int(input("Masukkan usia anda: "))
ValueError: invalid literal for int() with base 10: 'dua puluh'
yuanlukito@linuxstation:~/Documents$ █
```

Gambar 3.3: Jika input tidak sesuai yang diharapkan, program akan berhenti.

Bagaimana cara menangani input yang tidak sesuai? Salah satu cara yang dapat digunakan adalah menggunakan **try** dan **except**. Penggunaannya dalam kasus kategori usia dapat dilihat pada source code program berikut ini:

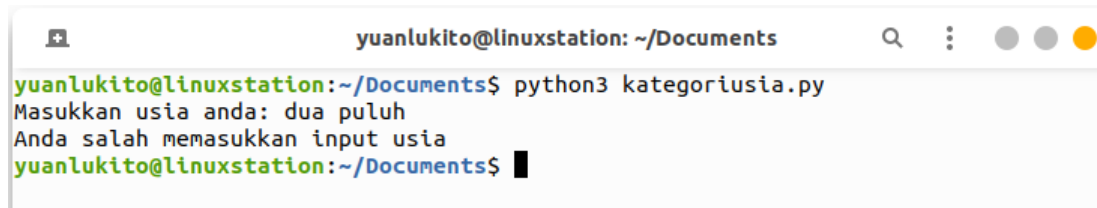
```
1 inputuser = input("Masukkan usia anda: ")
2 try:
3     usia = int(inputuser)
4     if usia <= 5:
5         print("Balita")
6     elif usia >= 6 and usia <= 11:
7         print("Kanak-kanak")
8     elif usia >=12 and usia <= 25:
9         print("Remaja")
10    elif usia >= 26 and usia <= 45:
```

```

11         print("Dewasa")
12     elif usia > 45:
13         print("Lansia")
14 except:
15     print("Anda salah memasukkan input usia")

```

Jika dijalankan dan diberi input yang tidak sesuai, program tidak mengalami kesalahan seperti sebelumnya. Hasilnya jika dijalankan dapat dilihat pada Gambar 3.4.



```

yuanlukito@linuxstation: ~/Documents
yuanlukito@linuxstation:~/Documents$ python3 kategoriusia.py
Masukkan usia anda: dua puluh
Anda salah memasukkan input usia
yuanlukito@linuxstation:~/Documents$

```

Gambar 3.4: Hasil program kategori usia.

3.4 Kegiatan Praktikum

Kegiatan praktikum yang akan dilakukan adalah sebagai berikut:

1. Studi kasus masalah-masalah yang membutuhkan percabangan (conditional).
2. Studi kasus penanganan input dari pengguna dan penggunaan try-except.

3.4.1 Contoh Masalah-masalah Percabangan

■ Contoh 3.1 Demam atau tidak?

Buatlah sebuah program yang meminta input suhu tubuh dari pengguna, kemudian program akan menentukan apakah pengguna mengalami demam atau tidak. Kriteria demam jika suhu tubuh lebih besar atau sama dengan 38 derajat Celcius. Alur dari program ini dapat dinyatakan dalam potongan flowchart seperti pada Gambar 3.5.

Hal pertama yang perlu anda pikirkan adalah apa input/masukan yang diperlukan untuk masalah ini? Dari deskripsi masalah kita bisa menentukan bahwa dibutuhkan informasi besarnya suhu tubuh untuk bisa menentukan apakah seseorang mengalami demam atau tidak. Masalah demam atau tidak ini memiliki dua kemungkinan hasil, yaitu demam ($\text{suhu} \geq 38$) dan tidak demam. Berarti kriteria untuk demam sudah diketahui yaitu $\text{suhu} \geq 38$. Bagaimana dengan kriteria untuk tidak demam? Karena kemungkinan hasilnya cuma dua, maka kriteria tidak demam merupakan kebalikan dari kriteria demam, yaitu $\text{suhu} < 38$. Cara lain untuk menyusun kriterianya adalah dengan cara menuliskan nilai-nilai yang memenuhi, seperti berikut:

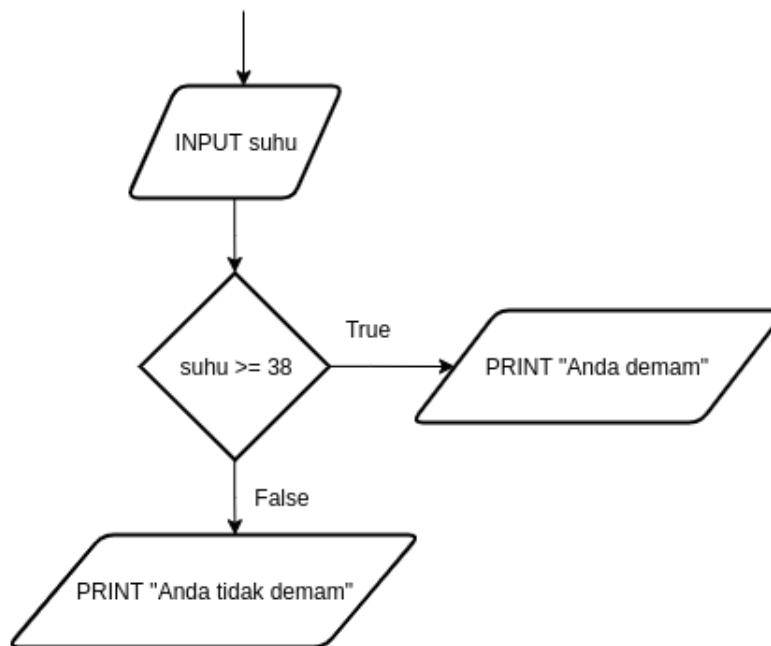
- Demam: 38, 39, 40, 41, ... (dan seterusnya). Dapat dinyatakan dalam bentuk $\text{suhu} \geq 38$.
- Tidak demam: 37, 36, 35, ... (dan seterusnya). Dapat dinyatakan dalam bentuk $\text{suhu} < 38$. Bisa juga dalam bentuk $\text{suhu} \leq 37$.

Program untuk memecahkan masalah ini adalah sebagai berikut:

```

1 suhu = int(input("Masukkan suhu tubuh: "))
2 if suhu >= 38:
3     print("Anda demam")

```



Gambar 3.5: Menentukan demam atau tidak berdasarkan suhu tubuh.

```
4 else:
5     print("Anda tidak demam")
```

Program tersebut jika dijalankan dan dicoba dengan beberapa input, hasilnya dapat dilihat pada Gambar 3.6.

The screenshot shows a terminal window with the title 'yuanlukito@linuxstation: ~/Documents'. The prompt is 'yuanlukito@linuxstation:~/Documents\$'. The first command is 'python3 demam.py', followed by the input 'Masukkan suhu tubuh: 42' and the output 'Anda demam'. The second command is 'python3 demam.py', followed by the input 'Masukkan suhu tubuh: 28' and the output 'Anda tidak demam'. The prompt is then 'yuanlukito@linuxstation:~/Documents\$' with a cursor.

Gambar 3.6: Menentukan demam atau tidak berdasarkan suhu tubuh.

■ Contoh 3.2 Positif atau Negatif?

Buat sebuah program yang dapat menentukan apakah bilangan yang dimasukkan oleh pengguna merupakan bilangan positif, bilangan negatif atau nol. Contoh input dan output yang diharapkan dapat dilihat pada Tabel 3.3.

Jika anda perhatikan kemungkinan hasil yang harus ditangani, maka ada tiga kemungkinan: positif, negatif dan nol. Kriteria untuk masing-masing kemungkinan tersebut adalah:

- Positif jika input bilangan > 0
- Negatif jika input bilangan < 0

Tabel 3.3: Contoh input dan output yang diharapkan.

Input	Output yang diharapkan
28	Positif
-90	Negatif
2000	Positif
0	Nol

- Nol jika input bilangan == 0

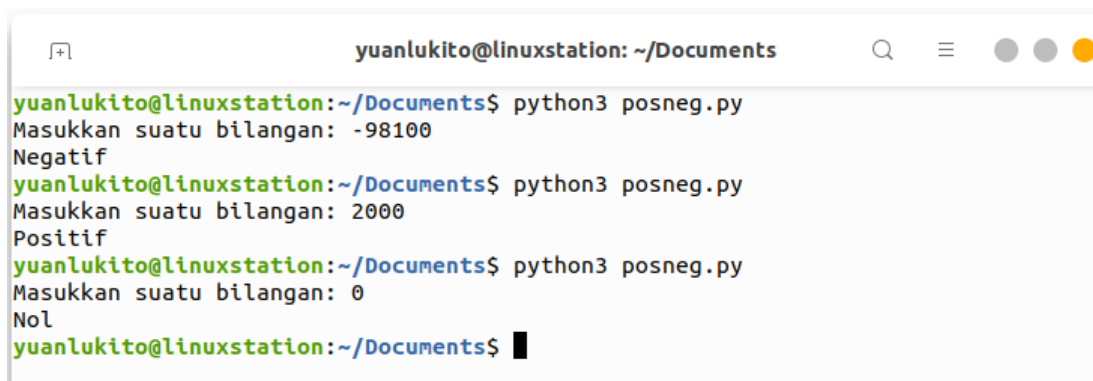
Setelah anda berhasil menyusun kemungkinan-kemungkinan dan semua kriterianya, anda sudah bisa menyusun programnya seperti contoh berikut ini:

```

1  bilangan = int(input("Masukkan suatu bilangan: "))
2  if bilangan > 0:
3      print("Positif")
4  elif bilangan < 0:
5      print("Negatif")
6  elif bilangan == 0:
7      print("Nol")

```

Jika program tersebut dijalankan dan diberikan beberapa input yang berbeda, hasilnya sudah sesuai dengan yang diharapkan seperti yang dapat anda lihat pada Gambar 3.7.



```

yuanlukito@linuxstation: ~/Documents
yuanlukito@linuxstation:~/Documents$ python3 posneg.py
Masukkan suatu bilangan: -98100
Negatif
yuanlukito@linuxstation:~/Documents$ python3 posneg.py
Masukkan suatu bilangan: 2000
Positif
yuanlukito@linuxstation:~/Documents$ python3 posneg.py
Masukkan suatu bilangan: 0
Nol
yuanlukito@linuxstation:~/Documents$

```

Gambar 3.7: Hasil dari program Positif-Negatif.

■ Contoh 3.3 Nilai Terbesar

Buat sebuah program yang meminta tiga input bilangan dari pengguna, kemudian program akan menampilkan bilangan yang terbesar dari ketiga input bilangan tersebut.

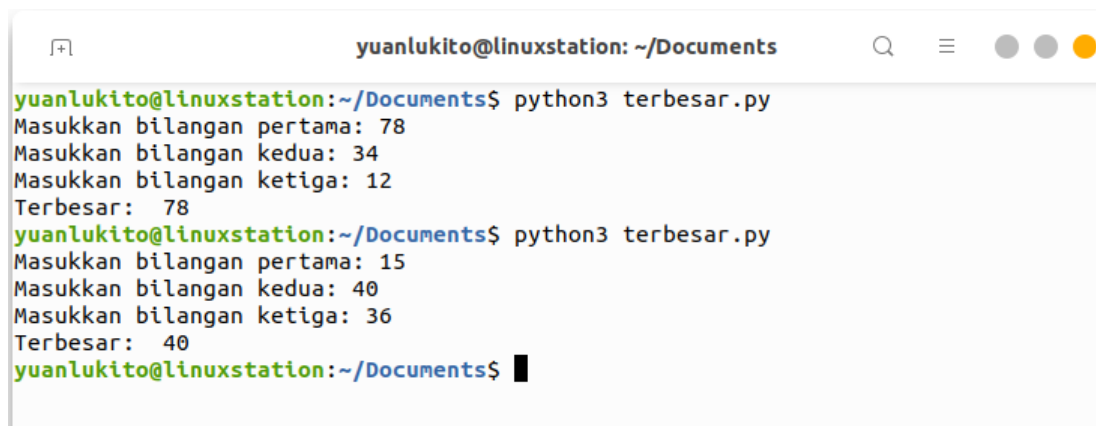
Untuk menangani masalah ini, kita andaikan terlebih dahulu ketiga bilangan tersebut adalah a, b dan c. Bilangan yang terbesar berarti ada tiga kemungkinan, yaitu bisa a, bisa b atau bisa c. Dari mana kita tahu bahwa a adalah yang terbesar? Kriterianya adalah sebagai berikut:

- A adalah yang terbesar jika $a > b$ dan $a > c$.
- B adalah yang terbesar jika $b > a$ dan $b > c$.
- C adalah yang terbesar jika $c > a$ dan $c > b$.

Setelah kita berhasil menentukan ketiga kriteria tersebut, maka kita dapat menyusun programnya sebagai berikut:

```
1 # input a, b dan c
2 a = int(input("Masukkan bilangan pertama: "))
3 b = int(input("Masukkan bilangan kedua: "))
4 c = int(input("Masukkan bilangan ketiga: "))
5
6 # secara berurutan tulis kriteria untuk a, b, dan c
7 if a > b and a > c:
8     print("Terbesar: ", a)
9 elif b > a and b > c:
10    print("Terbesar: ", b)
11 elif c > a and c > b:
12    print("Terbesar: ", c)
```

Jika program tersebut dijalankan dan dimasukkan beberapa macam input yang berbeda, hasilnya sudah sesuai dengan yang diharapkan. Anda dapat melihat hasilnya pada Gambar 3.8



The screenshot shows a terminal window titled 'yuanlukito@linuxstation: ~/Documents'. It displays two runs of a Python script named 'terbesar.py'. In the first run, the user inputs 78, 34, and 12, and the program outputs 'Terbesar: 78'. In the second run, the user inputs 15, 40, and 36, and the program outputs 'Terbesar: 40'.

```
yuanlukito@linuxstation: ~/Documents
yuanlukito@linuxstation:~/Documents$ python3 terbesar.py
Masukkan bilangan pertama: 78
Masukkan bilangan kedua: 34
Masukkan bilangan ketiga: 12
Terbesar: 78
yuanlukito@linuxstation:~/Documents$ python3 terbesar.py
Masukkan bilangan pertama: 15
Masukkan bilangan kedua: 40
Masukkan bilangan ketiga: 36
Terbesar: 40
yuanlukito@linuxstation:~/Documents$
```

Gambar 3.8: Hasil dari program mencari bilangan terbesar.

3.5 Latihan Mandiri

Latihan 3.1 Implementasikan penanganan kesalahan input pengguna dari program-program pada Contoh 3.1, 3.2 dan 3.3.

Latihan 3.2 Implementasikan percabangan pada Contoh 3.2 (Positif-Negatif) menggunakan ternary operator.

Latihan 3.3 Buatlah sebuah program yang dapat menampilkan jumlah hari dalam suatu bulan di tahun 2020. Program meminta pengguna memasukkan nomor bulan (1-12), kemudian program akan menampilkan jumlah hari pada bulan tersebut. Sebagai contoh, perhatikan input dan output berikut ini:

```
Masukkan bulan (1-12): 7
Jumlah hari: 31
```

Lengkapi program tersebut dengan penanganan kesalahan jika pengguna memasukkan bulan yang salah. Penanganan kesalahan dalam bentuk memunculkan pesan bahwa bulan yang diinputkan oleh pengguna tersebut tidak valid.

Latihan 3.4 Sebuah program meminta pengguna memasukkan ketiga panjang sisi suatu segitiga (berarti pengguna memasukkan tiga bilangan). Jika ketiga sisi segitiga tersebut semuanya sama, tampilkan pesan: "3 sisi sama". Jika hanya ada dua sisi yang sama panjang, tampilkan pesan "2 sisi sama". Jika tidak ada yang sama maka tampilkan pesan: "Tidak ada yang sama". Sebagai contoh, perhatikan input dan output berikut ini:

```
Masukkan sisi 1: 14
Masukkan sisi 2: 18
Masukkan sisi 3: 11
Tidak ada yang sama
```

```
Masukkan sisi 1: 22
Masukkan sisi 2: 22
Masukkan sisi 3: 22
3 sisi sama
```

```
Masukkan sisi 1: 8
Masukkan sisi 2: 9
Masukkan sisi 3: 8
2 sisi sama
```

Lengkapi program tersebut dengan penanganan kesalahan jika pengguna memasukkan input yang tidak valid.

4. Modular Programming

4.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat memahami anatomi dan struktur dari suatu fungsi.
2. Dapat memahami penggunaan parameter dan hasil dari suatu fungsi.
3. Dapat mendefinisikan fungsi dan parameter yang dibutuhkan dalam memecahkan suatu masalah.

4.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

4.3 Materi

4.3.1 Fungsi, Argument dan Parameter

Anda tentunya sudah mengerti tentang apa yang dilakukan oleh kode program berikut ini:

```
nama = input("Masukkan nama: ")
```

```
print("Hallo ", nama, " selamat pagi!")
```

Program tersebut meminta pengguna untuk memasukkan nama, kemudian program akan menyapa nama yang diinputkan oleh pengguna. Pada program tersebut ada dua fungsi yang digunakan, yaitu **input()** dan **print()**. Keduanya merupakan fungsi bawaan dari Python (atau biasa disebut sebagai *built-in function*). Setiap fungsi memiliki kegunaan masing-masing, seperti pada contoh tersebut fungsi **input()** digunakan untuk membaca input yang diberikan oleh pengguna, sedangkan fungsi **print()** digunakan untuk menampilkan tulisan di layar.

Secara umum fungsi adalah kumpulan perintah-perintah yang dijadikan satu, memiliki suatu tujuan dan kegunaan khusus serta dapat digunakan ulang. Apa kaitan fungsi dengan modular programming? Jika anda membuat program yang membutuhkan langkah yang banyak, anda akan membutuhkan untuk mengelompokkan beberapa kode program menjadi bagian-bagian (block) dari suatu program yang besar. Karena itu disebut sebagai modular, di mana program anda terdiri dari beberapa bagian modular yang memiliki kegunaan khusus dan dapat digunakan ulang. Berdasarkan asalnya, fungsi dibagi menjadi dua jenis yaitu:

- Fungsi bawaan (built-in function). Daftar fungsi bawaan Python 3 dapat dilihat di <https://docs.python.org/3/library/functions.html>
- Fungsi yang dibuat sendiri oleh programmer.

Sebagai contoh, perhatikan fungsi **tambah()** berikut ini yang dapat digunakan untuk menghitung jumlah dari dua bilangan yang diberikan:

```
def tambah(a, b):  
    hasil = a + b  
    return hasil
```

Fungsi **tambah** tersebut terdiri dari beberapa hal yang perlu diperhatikan:

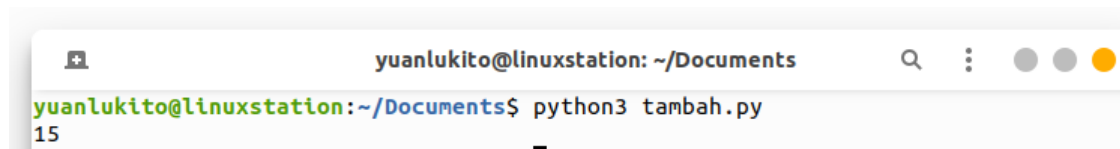
- Keyword **def** digunakan untuk mendefinisikan sebuah fungsi.
- Nama fungsi yang dibuat adalah **tambah()**.
- Isi dari fungsi harus anda tuliskan menjorok ke dalam 1 tab. Perhatikan bagian **tambah(a,b):** sebagai penanda blok.
- Fungsi **tambah()** membutuhkan dua argument, yang nantinya akan dikenali sebagai parameter **a** dan **b**.
- Fungsi tersebut akan menghasilkan hasil penjumlahan yang dapat ditampung di sebuah variabel. Keyword **return** digunakan untuk mengembalikan/mengeluarkan nilai dari suatu fungsi.

Penggunaan fungsi tersebut dapat dilihat pada kode program berikut ini:

```
1  # definisikan fungsi tambah terlebih dahulu  
2  def tambah(a, b):  
3      hasil = a + b  
4      return hasil  
5  
6  # panggil fungsi tambah dengan dua arguments berupa nilai: 10 dan 5  
7  c = tambah(10, 5)  
8  print(c)  
9
```

Jalannya program tersebut adalah sebagai berikut:

- Baris 1 adalah komentar, akan diabaikan oleh interpreter Python.
- Baris 2-4 adalah mendefinisikan fungsi tambah(). Baris 2-4 tidak dijalankan sampai suatu saat fungsi tambah() dipanggil.
- Baris 5-6 adalah baris kosong dan baris komentar akan diabaikan oleh Python.
- Baris 7 variabel c diisi oleh nilai yang dihasilkan oleh pemanggilan fungsi tambah(). Fungsi tambah dipanggil dengan argument 10 dan 5 (sesuai dengan urutan).
- Program akan lompat ke baris 2 karena fungsi tambah() dipanggil. Pada baris 2 terdapat parameter a dan b. Karena fungsi tambah() dipanggil dengan arguments 10 dan 5, maka parameter a = 10 dan b = 5 (sesuai urutan).
- Program lanjut ke baris 3. Pada baris ini variabel hasil akan berisi $10 + 5 = 15$.
- Program lanjut ke baris 4. Ada penggunaan keyword return, diikuti oleh variabel hasil yang nilainya 15. Return di sini menandakan bahwa fungsi tambah() sudah berakhir dan menghasilkan nilai 15 (sesuai dengan variabel hasil).
- Dari baris 4, program akan kembali ke baris 7. Ingat, jika fungsi sudah selesai, program akan kembali ke baris di mana fungsi tersebut dipanggil. Pada baris 7 sekarang nilai c diisi oleh 15 (hasil dari menjalankan fungsi tambah()).
- Pada baris 8 tampilkan nilai dari variabel c ke layar. Output yang dihasilkan adalah 15 seperti yang ditunjukkan pada Gambar 4.1.
- Jadi urutan jalannya program tersebut secara berturut-turut adalah baris 1-2-3-4-5-6-7-2-3-4-7-8.



Gambar 4.1: Hasil dari pemanggilan fungsi tambah().

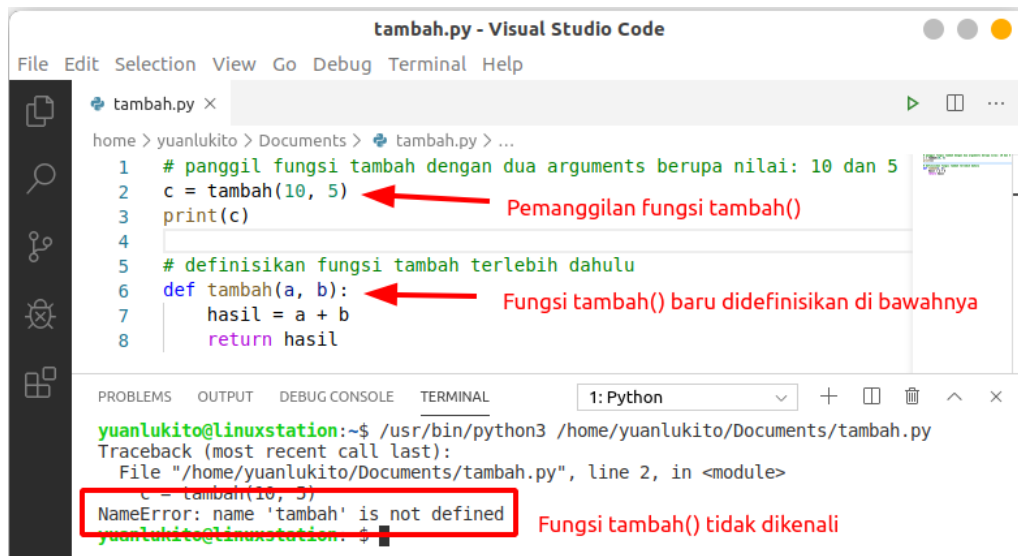
Fungsi dapat dipanggil jika sudah didefinisikan sebelumnya. Jika anda memanggil fungsi yang belum didefinisikan, atau jika fungsi tersebut didefinisikan di bawah, program anda akan mengalami kesalahan. Contoh pemanggilan fungsi sebelum fungsi didefinisikan dapat dilihat pada Gambar 4.2.

4.3.2 Return Value

Berdasarkan hasil yang dikeluarkan oleh fungsi, secara umum ada dua jenis yaitu: (1) fungsi yang tidak mengembalikan nilai dan (2) fungsi yang mengembalikan nilai. Fungsi yang tidak mengembalikan nilai sering disebut sebagai **void function**. Sebagai contoh, fungsi print_twice() berikut ini adalah fungsi yang tidak mengembalikan nilai:

```
def print_twice(message):  
    print(message)  
    print(message)  
  
print_twice("Hello World!")
```

Fungsi print_twice() membutuhkan 1 parameter yaitu message. Kemudian fungsi print_twice() akan menampilkan nilai dari variabel message sebanyak 2 kali. Fungsi print_twice() tidak menghasilkan suatu nilai yang dapat digunakan untuk proses berikutnya. Fungsi print_twice() sebenarnya mengembalikan nilai **None** jika anda coba panggil dengan cara berikut:



Gambar 4.2: Fungsi tambah() belum dikenali karena didefinisikan di bawahnya.

```
def print_twice(message):
    print(message)
    print(message)

print(print_twice("Hello World!"))
```

Output yang dihasilkan adalah:

```
Hello World!
Hello World!
None
```

Berbeda halnya dengan fungsi tambah() berikut ini:

```
def tambah(a, b, c):
    hasil = a + b + c
    return hasil
```

Fungsi tambah() membutuhkan 3 parameter yaitu a, b dan c. Kemudian di dalam fungsi tambah() didefinisikan suatu variabel hasil yang diisi oleh a+b+c. Kemudian variabel hasil tersebut dikeluarkan dari fungsi sebagai hasil dari fungsi tambah() dengan keyword **return**. Keyword return digunakan untuk:

- Mengeluarkan nilai yang merupakan hasil dari fungsi.
- Mengakhiri fungsi.

Fungsi tambah() tersebut bisa digunakan untuk mencari hasil penjumlahan tiga bilangan. Hasil dari penjumlahan tiga bilangan tersebut dapat digunakan untuk proses/langkah berikutnya, seperti contoh program untuk mencari rata-rata dari tiga bilangan berikut ini:

```
1 def tambah(a, b, c):
2     hasil = a + b + c
```

```

3     return hasil
4
5     nilai1 = 70
6     nilai2 = 85
7     nilai3 = 55
8
9     rata_rata = tambah(nilai1, nilai2, nilai3)/3
10    print(rata_rata)

```

Urutan jalannya program tersebut adalah sebagai berikut:

- Baris 1, 2, 3 adalah definisi fungsi tambah. Belum ada yang dijalankan.
- Baris 5, 6, 7 mendefinisikan tiga variabel dan langsung diisi nilainya.
- Baris 9 memanggil fungsi tambah, dengan mengirimkan tiga arguments yaitu nilai1 (70), nilai2 (85) dan nilai 3 (55). Program akan berjalan ke baris 1.
- Baris 1 ada 3 parameter yang diisi nilainya oleh tiga arguments yang dikirimkan, sehingga $a = 70$, $b = 85$ dan $c = 55$.
- Baris 2 mendefinisikan variabel hasil yang diisi oleh nilai $a+b+c$, sehingga $hasil = 70 + 85 + 55 = 210$.
- Baris 3 return nilai dari hasil, berarti fungsi tambah() mengeluarkan hasil dengan nilai 210. Dengan adanya return, artinya fungsi tambah() sudah berakhir dan program akan berjalan ke baris 9 (baris di mana fungsi tambah() dipanggil sebelumnya).
- Baris 9 sudah didapatkan hasil dari fungsi tambah, sehingga menjadi $rata_rata = 210/3 = 70$. Variabel rata_rata bernilai 70.
- Baris 10 menampilkan nilai dari variabel rata_rata, yaitu 70.

4.3.3 Optional Argument dan Named Argument

Fungsi dapat memiliki optional parameter, yaitu parameter yang bersifat opsional dan memiliki nilai bawaan (default) yang sudah didefinisikan sebelumnya. Untuk mendefinisikan optional parameter, anda harus mendefinisikan nilai bawaannya terlebih dahulu seperti pada contoh berikut ini:

```

def hitung_belanja(belanja, diskon=0):
    bayar = belanja - (belanja * diskon)/100
    return bayar

```

Fungsi hitung_belanja() memiliki dua parameter yaitu **belanja** dan **diskon**. Parameter diskon secara default memiliki nilai 0 (yang artinya 0%). Untuk memanggil fungsi hitung_belanja() tersebut, anda dapat melakukan dengan beberapa cara seperti berikut ini:

```

def hitung_belanja(belanja, diskon=0):
    bayar = belanja - (belanja * diskon)/100
    return bayar

print(hitung_belanja(100000))
print(hitung_belanja(100000, 10))
print(hitung_belanja(100000, 50))

```

Output yang dihasilkan adalah sebagai berikut:

```
100000.0
90000.0
50000.0
```

Pemanggilan pertama hanya menggunakan satu argument, sedangkan pemanggilan kedua dan ketiga menggunakan dua argument.

Setiap parameter pada fungsi memiliki nama. Karena itu pada pemanggilan fungsi juga dapat disertakan nama parameternya dan tidak perlu sesuai dengan urutan yang diberikan. Perhatikan contoh berikut ini:

```
def cetak(a, b, c):
    print("Nilai a: ",a)
    print("Nilai b: ",b)
    print("Nilai c: ",c)

cetak(20, 30, 40)
```

Output yang dihasilkan adalah:

```
Nilai a: 20
Nilai b: 30
Nilai c: 40
```

Anda juga dapat memanggil fungsi cetak() tersebut dengan cara berikut ini:

```
def cetak(a, b, c):
    print("Nilai a: ",a)
    print("Nilai b: ",b)
    print("Nilai c: ",c)

cetak(a=20, b=30, c=40)
```

Pemanggilan fungsi cetak() dilakukan dengan menyebutkan nama argumentnya (named argument). Jika anda menggunakan cara tersebut, maka urutan argument tidak harus sama dengan urutan parameter pada fungsi, seperti pada contoh berikut ini:

```
def cetak(a, b, c):
    print("Nilai a: ",a)
    print("Nilai b: ",b)
    print("Nilai c: ",c)

cetak(b=30, c=40, a=20)
```

4.3.4 Anonymous Function (Lambda)

Sesuai dengan namanya, anonymous function adalah fungsi tanpa nama (anonymous). Anonymous function pada Python adalah fitur tambahan, bukan merupakan fitur utama. Berbeda dengan bahasa-bahasa pemrograman seperti Haskell, Lisp dan Erlang yang merupakan bahasa pemrograman fungsional. Pada Python, digunakan keyword lambda untuk mendefinisikan anonymous function. Sebagai contoh, perhatikan fungsi tambah() berikut ini:

```
def tambah(a, b):  
    hasil = a + b  
    return hasil  
  
print(tambah(10,20))
```

Untuk mendefinisikan fungsi tambah menggunakan lambda adalah sebagai berikut:

```
tambah = lambda a, b: a + b  
  
print(tambah(10,20))
```

Setiap anonymous function pada Python terdiri dari beberapa bagian berikut ini:

- Keyword: lambda
- Bound variable: argument pada lambda function
- Body: bagian utama lambda, berisi ekspresi atau statement yang menghasilkan suatu nilai.

4.4 Kegiatan Praktikum

Kegiatan praktikum yang akan dilakukan adalah sebagai berikut:

1. Mendefinisikan fungsi sesuai dengan spesifikasi yang dibutuhkan.
2. Penggunaan optional argument dan named argument.
3. Mendefinisikan dan menggunakan lambda function

4.4.1 Mendefinisikan Fungsi

■ Contoh 4.1 Fungsi Menghitung Tagihan Listrik

Buatlah sebuah fungsi yang dapat menghitung tagihan listrik seseorang (pasca bayar) dengan beberapa informasi berikut ini:

- Jumlah pemakaian (dalam kwh)
- Golongan tarif (1 - 4). Diasumsikan input golongan tarif selalu valid.
 - Golongan 1: Rp. 1500/kwh untuk 100 kwh pertama, selanjutnya dikenakan Rp. 2000/kwh.
 - Golongan 2: Rp. 2500/kwh untuk 100 kwh pertama, selanjutnya dikenakan Rp. 3000/kwh.
 - Golongan 3: Rp. 4000/kwh untuk 100 kwh pertama, selanjutnya dikenakan Rp. 5000/kwh.
 - Golongan 4: Rp. 5000/kwh untuk 100 kwh pertama, selanjutnya dikenakan Rp. 7000/kwh.

Jika tidak ada informasi golongan tarif, maka diasumsikan menggunakan tarif golongan 3. Definisi fungsi tersebut! ■

Ada beberapa hal yang perlu diperhatikan sebelum anda mendefinisikan fungsi untuk menghitung tagihan listrik tersebut:

- Fungsi membutuhkan dua parameter, yaitu jumlah pemakaian dan golongan tarif.
- Jumlah pemakaian menggunakan tarif yang berbeda untuk 100 kwh pertama, dan setelah 100 kwh.
- Golongan tarif memiliki nilai default = 3, sehingga harus didefinisikan sebagai optional argument.

Fungsi untuk menghitung tagihan listrik dan contoh penggunaannya dapat dilihat pada kode program berikut ini:

```

1 def tagihan_listrik(pemakaian, golongan=3):
2     bayar = 0
3     pemakaian_100 = 100 if pemakaian > 100 else pemakaian
4     pemakaian_100_lebih = pemakaian - pemakaian_100
5     if golongan == 1:
6         bayar = pemakaian_100 * 1500 + pemakaian_100_lebih * 2000
7     elif golongan == 2:
8         bayar = pemakaian_100 * 2500 + pemakaian_100_lebih * 3000
9     elif golongan == 3:
10        bayar = pemakaian_100 * 4000 + pemakaian_100_lebih * 5000
11    elif golongan == 4:
12        bayar = pemakaian_100 * 5000 + pemakaian_100_lebih * 7000
13    return bayar
14
15 print(tagihan_listrik(130))
16 print(tagihan_listrik(80, 4))
17 print(tagihan_listrik(golongan=1, pemakaian=175))

```



Pada kode program tersebut ada beberapa konsep yang diterapkan: optional argument, named argument dan ternary operator.

4.4.2 Argument dan Parameter

■ Contoh 4.2 Pengiriman Argument pada Fungsi

Perhatikan definisi fungsi `abc()` dan penggunaannya berikut ini:

```

def abc(a, b, c):
    a = b + c
    b = c + a
    c = a + b

nilai1 = 20
nilai2 = 30
nilai3 = 40
abc(nilai1, nilai2, nilai3)
print(nilai1)
print(nilai2)
print(nilai3)

```

Apa output yang dihasilkan dari kode program tersebut? Jelaskan! ■

Pengiriman parameter pada Python menggunakan prinsip yang disebut sebagai **Call-by-Object**. Perlakuan terhadap parameter tergantung dari apakah argument yang diberikan immutable atau tidak. Jika argument yang dikirimkan bersifat immutable artinya argument tersebut tidak bisa diubah oleh fungsi. Tipe data seperti integer, string dan tuple bersifat immutable.

Pada fungsi `abc()` dikirimkan 3 argument integer, yaitu `nilai1`, `nilai2` dan `nilai3`. Karena itu dapat dikatakan argument yang dikirimkan akan bersifat immutable, sehingga tidak akan terpengaruh nilainya oleh perubahan nilai parameter dalam fungsi `abc()`. Output yang dihasilkan saat program tersebut dijalankan adalah sebagai berikut:

20
30
40

4.4.3 Anonymous/Lambda Function

■ Contoh 4.3 Konversi ke Lambda Function

Perhatikan fungsi yang dapat menentukan apakah suatu bilangan merupakan kelipatan 9 atau tidak berikut ini:

```
def kelipatan_sembilan(angka):  
    if angka % 9 == 0:  
        return True  
    else:  
        return False
```

Ubahlah fungsi tersebut menjadi bentuk lambda function!

■

Setiap lambda function pada Python terdiri dari beberapa bagian. Dalam kasus ini, bagian-bagian tersebut adalah:

- Keyword: lambda
- Bound variable: angka
- Body: pengecekan apakah habis dibagi 9 atau tidak (kelipatan 9)

Sehingga bentuk lambda function-nya adalah sebagai berikut:

```
kelipatan_sembilan = lambda angka: angka % 9 == 0
```

Contoh penggunaan lambda function tersebut adalah sebagai berikut:

```
kelipatan_sembilan = lambda angka: angka % 9 == 0
```

```
print(kelipatan_sembilan(81))  
print(kelipatan_sembilan(2000))
```

4.5 Latihan Mandiri

Latihan 4.1 Buatlah sebuah fungsi yang dapat menentukan apakah ketiga parameter memenuhi semua ketentuan berikut ini:

- Ketiga parameter tersebut nilainya berbeda semua.
- Ada kemungkinan jika diambil dua parameter dan dijumlahkan hasilnya sama dengan parameter lainnya (yang tersisa).

Fungsi tersebut akan menghasilkan nilai True jika semua ketentuan tersebut dipenuhi. Jika tidak terpenuhi maka fungsi akan menghasilkan nilai False. Fungsi anda harus diberi nama cek_angka().

■

Latihan 4.2 Buatlah sebuah fungsi yang dapat menentukan apakah **minimal dua** dari tiga parameter yang diberikan memiliki digit paling kanan yang sama. Fungsi tersebut menghasilkan nilai True jika memenuhi dan False jika tidak memenuhi. Gunakan fungsi tersebut untuk mengecek beberapa test-case berikut ini:

- Input = 30, 20, 18. Output yang diharapkan = True
- Input = 145, 5, 100. Output yang diharapkan = True
- Input = 71, 187, 18. Output yang diharapkan = False
- Input = 1024, 14, 94. Output yang diharapkan = True
- Input = 53, 8900, 658. Output yang diharapkan = False

Ketiga bilangan tersebut diinputkan oleh pengguna, sehingga anda perlu membaca input dari pengguna. Fungsi anda harus diberi nama `cek_digit_belakang()`. ■

Latihan 4.3 Buatlah fungsi-fungsi konversi suhu menggunakan lambda function. Fungsi-fungsi yang harus anda implementasikan:

- Celcius to Fahrenheit. $F = (9/5) * C + 32$
- Celcius to Reamur. $R = 0.8 * C$

Berikan contoh penggunaannya untuk test-case berikut ini:

- Input C = 100. Output F = 212.
 - Input C = 80. Output R = 64.
 - Input = 0. Output F = 32.
-

5. Struktur Kontrol Perulangan

5.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menggunakan bentuk perulangan for maupun while.
2. Dapat menggunakan break dan continue sesuai dengan permasalahan yang dihadapi.
3. Dapat melakukan konversi dari bentuk for menjadi bentuk while
4. Dapat menerapkan perulangan dalam menyelesaikan permasalahan.

5.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

5.3 Materi

5.3.1 Definisi Perulangan

Jalannya suatu program dapat diatur secara sekuensial, percabangan, perulangan, maupun kombinasi dari ketiganya. Pengaturan tersebut biasa disebut sebagai struktur kontrol. Perulangan digunakan apabila dalam program diperlukan untuk:

- Melakukan suatu hal yang sama beberapa kali.

- Melakukan suatu hal secara bertahap, di mana setiap tahap sebenarnya memiliki langkah yang sama.
- Mengakses sekumpulan data dalam suatu struktur data, misalnya: List, Tuple, Queue, Stack dan beberapa struktur data lainnya.

Pada Python, perulangan dapat dilakukan dengan menggunakan **for**, **while** maupun dengan cara **rekursif**. Untuk pertemuan ini, dibahas mengenai perulangan for dan while.

5.3.2 Bentuk Perulangan for

Pada Python, perulangan dapat dinyatakan dalam bentuk **for** dan **while**. Perulangan **for** biasanya digunakan pada kondisi:

- **Jumlah perulangan sudah diketahui sejak awal.** Misalnya akan dilakukan pembacaan data dari 10 file teks. Walaupun setiap file teks memiliki isi yang berbeda, tetapi membaca file teks secara umum tetap sama. Pembacaan akan dilakukan dari file pertama, kedua, ketiga, dan seterusnya sampai file ke-sepuluh.
- **Perulangan terjadi karena operasi yang sama pada suatu rentang data atau rentang nilai.** Misalnya dalam mencari jumlah dari 100 bilangan pertama, maka secara berturut-turut dilakukan penjumlahan $1 + 2 + 3 + \dots$ <berulang-ulang> $+ 100$. Berarti dilakukan dalam rentang mulai dari 1 sampai 100.

Perulangan for pada rentang tertentu lebih mudah dilakukan dengan menggunakan bantuan fungsi `range()`, yang bentuknya sebagai berikut:

- `range(stop)`. Digunakan untuk menghasilkan rentang dari **0** sampai **stop-1**. Misalnya `range(6)`, berarti menghasilkan rentang 0-5.
- `range(start, stop, [step])`. Digunakan untuk menghasilkan rentang dari **start**, sampai **stop** dengan peningkatan sejumlah **step**.

Berikut ini adalah contoh program untuk menampilkan bilangan dari 1 sampai 100 dengan menggunakan for dan `range()`:

```
for i in range(1, 101):
    print(i)
```

Pada program tersebut, ada perulangan for dengan menggunakan `range()`, dimulai dari 1 (start) sampai 101 (stop-1) dengan langkah 1 (default step adalah 1). Kemudian variabel `i` digunakan sebagai counter, di mana nilai `i` akan naik secara berurutan sesuai dengan nilai yang dihasilkan dari fungsi `range()` tersebut. Untuk kasus di mana tidak diperlukan counter, maka bentuk perulangan bisa seperti berikut ini:

```
for _ in range(1, 101):
    print('Hello World')
```

Program tersebut akan menampilkan tulisan Hello World sebanyak 100 kali, yang tidak membutuhkan nilai dari suatu counter.

Step negatif

Perhatikan perulangan for berikut ini yang akan menampilkan seluruh bilangan genap dari 2 sampai 100:

```
for i in range(2, 101, 2):
    print(i)
```

Perulangan dilakukan pada rentang 2-100, dengan langkah 2. Maka rentang yang dipakai adalah 2, 4, 6, 8, 10, 12, ..., 100. Bagaimana jika diperlukan untuk menampilkan bilangan genap dari 100 sampai 2? Fungsi `range()` bisa menerima step negatif, seperti pada contoh berikut ini:

```
for i in range(100, 1, -2):
    print(i)
```

Program tersebut akan menampilkan bilangan genap mulai dari 100, 98, 96, 94, ..., sampai 2.

5.3.3 Bentuk Perulangan While

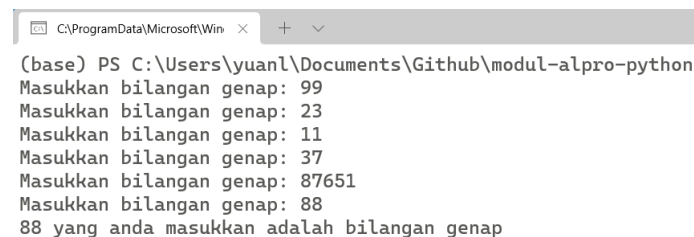
Bentuk while biasanya digunakan pada kondisi di mana jumlah perulangan belum diketahui sebelumnya. Bentuk perulangan while secara umum adalah sebagai berikut:

```
<start>
while <stop>:
    operation
    operation
    <step (optional)>
```

Berikut ini adalah contoh perulangan dengan menggunakan while yang digunakan untuk memastikan input yang dimasukkan oleh pengguna adalah bilangan genap:

```
1 bilangan = 0
2 genap = False
3 while genap == False:
4     bilangan = int(input('Masukkan bilangan genap: '))
5     if bilangan % 2 == 0:
6         genap = True
7 print(bilangan, 'yang anda masukkan adalah bilangan genap')
```

Output yang dihasilkan oleh program tersebut dapat dilihat pada Gambar 5.1. Pengguna awalnya memasukkan bilangan ganjil, tetapi program terus meminta pengguna memasukkan bilangan genap. Program berhenti setelah pengguna memasukkan 88, yang merupakan bilangan genap. Kasus ini sangat sesuai jika menggunakan perulangan while, karena tidak diketahui sampai berapa kali pengguna memasukkan bilangan ganjil yang tidak sesuai dengan permintaan.



```
(base) PS C:\Users\yuanl\Documents\Github\modul-alpro-python
Masukkan bilangan genap: 99
Masukkan bilangan genap: 23
Masukkan bilangan genap: 11
Masukkan bilangan genap: 37
Masukkan bilangan genap: 87651
Masukkan bilangan genap: 88
88 yang anda masukkan adalah bilangan genap
```

Gambar 5.1: Penggunaan while untuk mengambil input dari pengguna sampai sesuai dengan permintaan.

5.3.4 Penggunaan Break dan Continue

Perulangan dapat dikontrol dengan menggunakan **break** dan **continue**. Secara umum break digunakan untuk menghentikan perulangan, sedangkan continue digunakan untuk melanjutkan perulangan ke iterasi berikutnya. Perhatikan program berikut ini yang akan menampilkan bilangan dari 1 sampai 10:

```
for i in range(1, 11):
    print(i)
print('Selesai')
```

Program tersebut akan menampilkan bilangan 1 sampai 10, kemudian pada baris terakhir muncul tulisan 'Selesai'. Jika diinginkan perulangan yang seharusnya sampai 10, dihentikan saat mencapai 5, maka diperlukan break dengan kondisi seperti program berikut ini:

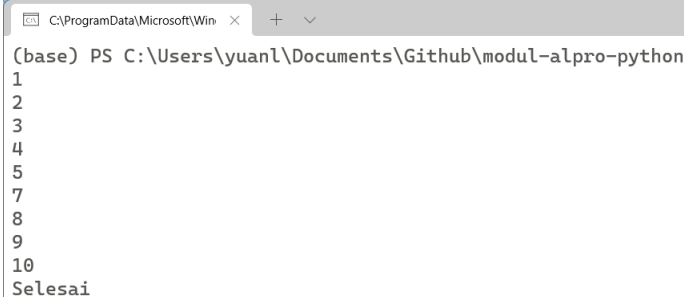
```
for i in range(1, 11):
    if i == 5:
        break
    else:
        print(i)
print('Selesai')
```

Pada saat nilai $i=5$, maka boolean expression dari $i==5$ akan menghasilkan nilai True, program akan menjalankan instruksi break. Maka selanjutnya perulangan akan dihentikan dan program berlanjut pada baris setelah perulangan tersebut, yaitu menampilkan tulisan 'Selesai'.

Apa perbedaan antara break dan continue? Continue digunakan pada saat diperlukan untuk melewati tahap perulangan sekarang, langsung dilanjutkan ke tahap/iterasi perulangan berikutnya. Sebagai contoh, berikut ini adalah program yang menampilkan angka dari 1 sampai 10, tetapi diperlukan untuk tidak menampilkan angka 6:

```
for i in range(1, 11):
    if i == 6:
        continue
    else:
        print(i)
print('Selesai')
```

Output yang dihasilkan dari contoh penggunaan continue dapat dilihat pada Gambar 5.2. Program menampilkan angka dari 1 sampai 10, tetapi melewati angka 6.



```
C:\ProgramData\Microsoft\Win...
(base) PS C:\Users\yuanl\Documents\Github\modul-alpro-python
1
2
3
4
5
7
8
9
10
Selesai
```

Gambar 5.2: Penggunaan continue untuk melewati iterasi saat counter(i) bernilai 6.

5.3.5 Konversi dari Bentuk for Menjadi Bentuk while

Bentuk perulangan for sebagian besar dapat dikonversi menjadi bentuk while. Beberapa hal yang ada di bentuk for dan while adalah sebagai berikut:

- Harus ada nilai awal, untuk memulai perulangan.
 - Harus ada nilai akhir, untuk mengakhiri perulangan.
 - Harus ada langkah, agar iterasi dari nilai awal bisa terus berjalan sampai mencapai nilai akhir.
- Misalkan ada perulangan for seperti berikut:

```
for i in range(1, 11):
    print(i)
```

maka dapat diidentifikasi bahwa perulangan tersebut dimulai dari 1, berakhir di 10, dengan step adalah 1. Konversi ke bentuk while dapat dilakukan dengan mudah, menghasilkan perulangan while berikut ini yang menghasilkan output yang sama:

```
i = 1           //awal
while i <= 10:  //kondisi akhir
    print(i)
    i = i + 1   //step
```

5.4 Kegiatan Praktikum

Kegiatan praktikum pada pertemuan ini adalah sebagai berikut:

- Menggunakan perulangan untuk menampilkan deret bilangan.
- Menggunakan break pada perulangan while.

5.4.1 Deret Bilangan

■ **Contoh 5.1** Buatlah program untuk menampilkan deret bilangan fibonacci mulai dari 1 sampai batas tertentu yang dimasukkan oleh pengguna!

Deret bilangan fibonacci adalah deret bilangan yang tersusun dari penjumlahan dua suku sebelumnya dari deret bilangan tersebut. Biasanya deret bilangan fibonacci dimulai dari 1, 1, 2, 3, ... dan seterusnya. Ilustrasinya dapat dilihat pada Gambar 5.3.

1 1 2 3 5 8 13 21 34 ...
 └─┬─┘
 13 + 21 = 34

Gambar 5.3: Deret bilangan fibonacci.

Implementasi dari deret fibonacci lebih baik jika dipisahkan dalam bentuk fungsi fibo() yang menerima parameter batas. Langkah-langkah dari implementasi tersebut adalah:

1. Minta nilai batas dari pengguna.
2. Panggil fungsi fibo() untuk menampilkan deret fibonacci dengan menggunakan perulangan while.

Program untuk menjawab permasalahan tersebut adalah sebagai berikut:

```
def fibo(batas):
    bil1 = 1
    bil2 = 1
    # tampilkan dua suku fibonacci pertama
    if bil1 < batas:
        print(bil1, end='\t')
        print(bil2, end='\t')
    # suku-suku berikutnya dari bil1 + bil2
    suku_baru = bil1 + bil2
    while suku_baru < batas:
        print(suku_baru, end='\t')
        # geser bil1 dan bil2
```

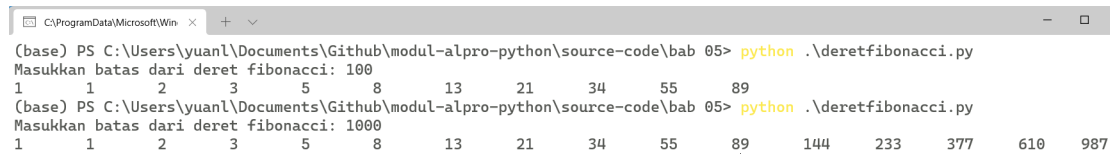
```

bil1 = bil2
bil2 = suku_baru
# hitung lagi suku berikutnya
suku_baru = bil1 + bil2

# program utama
batas = int(input('Masukkan batas dari deret fibonacci: '))
fibo(batas)

```

Jika program tersebut dijalankan, hasilnya akan seperti pada Gambar 5.4.



```

(base) PS C:\Users\yuanl\Documents\Github\modul-alpro-python\source-code\bab 05> python .\deretfibonacci.py
Masukkan batas dari deret fibonacci: 100
1      1      2      3      5      8      13      21      34      55      89
(base) PS C:\Users\yuanl\Documents\Github\modul-alpro-python\source-code\bab 05> python .\deretfibonacci.py
Masukkan batas dari deret fibonacci: 1000
1      1      2      3      5      8      13      21      34      55      89      144      233      377      610      987

```

Gambar 5.4: Deret bilangan fibonacci yang kurang dari 100 dan kurang dari 1000.

■ **Contoh 5.2** Buatlah program yang dapat menampilkan deret bilangan konvergen, yang diawali dari input bilangan dari pengguna. Suku-suku dari deret tersebut berikutnya didapatkan dengan cara berikut ini:

- Jika ganjil, maka kalikan dengan tiga, lalu tambah 1.
- Jika genap, bagi dengan 2.

Tampilkan suku-suku dari deret bilangan konvergen tersebut yang berakhir jika mencapai nilai 1. ■

Untuk menyelesaikan permasalahan ini, diperlukan pemahaman terlebih dahulu mengenai deret bilangan konvergen tersebut. Sebagai contoh bila input dari pengguna adalah 5, maka deret bilangan yang dihasilkan adalah: 5, 16, 8, 4, 2, 1. Bagaimana jika pengguna memberi input 12? Deret yang dihasilkan adalah: 12, 6, 3, 10, 5, 16, 8, 4, 2, 1. Dari contoh tersebut, apakah dapat diketahui berapa jumlah suku yang akan ditampilkan? Jika jumlah suku tidak diketahui dari awal, maka permasalahan ini lebih cocok jika diselesaikan menggunakan perulangan bentuk while.

Untuk menyusun perulangan bentuk while, diperlukan beberapa bagian berikut:

- Awal adalah input bilangan dari pengguna.
- Akhir adalah 1. Deret bilangan berakhir jika sudah mencapai nilai 1.
- Step dilakukan sesuai dengan suku sekarang, apakah genap atau ganjil.

Setelah mengetahui awal, akhir dan step, maka program untuk penyelesaian masalah tersebut adalah sebagai berikut:

```

def konvergen(start):
    suku = start
    while suku != 1:
        print(suku)
        if suku % 2 == 0:
            suku = suku // 2
        else:
            suku = suku * 3 + 1

# bagian utama program
start = int(input('Masukkan suku pertama dari deret konvergen: '))
konvergen(start)

```

Jika program tersebut dijalankan, hasilnya dapat dilihat pada Gambar 5.5.



```
C:\ProgramData\Microsoft\Win...
(base) PS C:\Users\yuan\Documents\Github\modul-alpro-python\source-code\bab 05> py
Masukkan suku pertama dari deret konvergen: 12
12
6
3
10
5
16
8
4
2
(base) PS C:\Users\yuan\Documents\Github\modul-alpro-python\source-code\bab 05> py
Masukkan suku pertama dari deret konvergen: 5
5
16
8
4
2
(base) PS C:\Users\yuan\Documents\Github\modul-alpro-python\source-code\bab 05> |
```

Gambar 5.5: Deret bilangan konvergen yang dimulai dari 12 dan 5.

5.4.2 Penggunaan Break

■ **Contoh 5.3** Buatlah program yang dapat menghitung rata-rata dari sejumlah input yang diberikan oleh pengguna. Program akan terus meminta input dari pengguna, sampai pengguna memasukkan bilangan negatif atau nol. Program kemudian menampilkan rata-rata dari keseluruhan input (abaikan input negatif atau nol). ■

Untuk menyelesaikan masalah ini, perlu diketahui terlebih dahulu bahwa program akan meminta input dari pengguna terus-menerus sampai pengguna memasukkan nilai negatif atau nol. Apakah dari awal sudah diketahui berapa kali pengguna memasukkan input sampai akhirnya memasukkan nilai negatif atau nol? Jawabannya adalah tidak diketahui. Oleh karena itu, untuk penyelesaian masalah ini akan menggunakan perulangan bentuk `while`. Untuk mengakhiri permintaan input dari pengguna, dapat dilakukan dengan penggunaan `break`, dengan kondisi input adalah negatif atau nol. Penyelesaian dari masalah tersebut adalah sebagai berikut:

```
def average():
    total = 0
    count = 0
    while True:
        input_user = int(input('Masukkan nilai (nol atau negatif untuk berhenti): '))
        if input_user < 1: # negatif atau nol
            break
        else:
            total = total + input_user
            count = count + 1
    if count > 0:
        return total / count
    else:
        return 0

# bagian utama program
hasil = average()
print('Rata-rata: ', hasil)
```

Contoh output dari program tersebut dapat dilihat pada Gambar 5.6.

Dari output yang dihasilkan terlihat bahwa program berhenti meminta input pengguna saat mendapatkan input bilangan negatif atau nol.

```

C:\ProgramData\Microsoft\Win
(base) PS C:\Users\yuanl\Documents\Github\modul-alpro-python\source-code\bab 05>
Masukkan nilai (nol atau negatif untuk berhenti): 18
Masukkan nilai (nol atau negatif untuk berhenti): 12
Masukkan nilai (nol atau negatif untuk berhenti): 30
Masukkan nilai (nol atau negatif untuk berhenti): -3
Rata-rata: 20.0
(base) PS C:\Users\yuanl\Documents\Github\modul-alpro-python\source-code\bab 05>
Masukkan nilai (nol atau negatif untuk berhenti): 70
Masukkan nilai (nol atau negatif untuk berhenti): 40
Masukkan nilai (nol atau negatif untuk berhenti): 100
Masukkan nilai (nol atau negatif untuk berhenti): 55
Masukkan nilai (nol atau negatif untuk berhenti): 82
Masukkan nilai (nol atau negatif untuk berhenti): 0
Rata-rata: 69.4
(base) PS C:\Users\yuanl\Documents\Github\modul-alpro-python\source-code\bab 05>

```

Gambar 5.6: Penggunaan break untuk menghentikan perulangan while.

5.5 Latihan Mandiri

Latihan 5.1 Buatlah program yang menerapkan perhitungan perkalian dengan menggunakan penjumlahan. Buatlah fungsi perkalian() dalam program tersebut! Berikut ini adalah beberapa contoh perhitungan yang diharapkan:

- $6 \times 5 = 5 + 5 + 5 + 5 + 5 + 5 = 30$.
- $7 \times 10 = 10 + 10 + 10 + 10 + 10 + 10 + 10 = 70$.

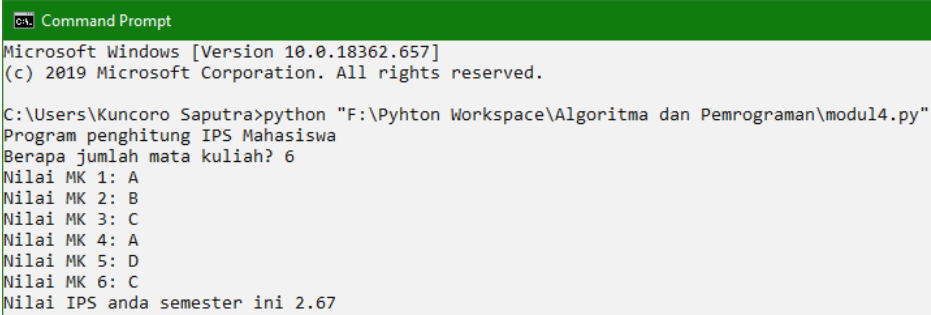
Latihan 5.2 Buatlah program yang dapat menampilkan deret bilangan ganjil dari batas bawah dan batas atas yang diberikan oleh pengguna. Jika ternyata batas atas < batas bawah, berarti deret tersebut dimulai dari batas atas, sampai batas bawah (negatif range). Buatlah fungsi ganjil() dalam program tersebut! Berikut ini adalah contoh hasil yang diharapkan:

- bawah = 10, atas = 30. Karena bawah < atas, berarti dari kecil ke besar, maka hasilnya adalah: 11, 13, 15, 17, 19, 21, 23, 25, 27, 29.
- bawah = 97, atas = 82. Karena bawah > atas, berarti dari besar ke kecil, maka hasilnya adalah: 97, 95, 93, 91, 89, 87, 85, 83.

Latihan 5.3 Buatlah sebuah program penghitung nilai Indeks Prestasi Semester (IPS). Input bagi program:

- Jumlah mata kuliah
- Nilai A, B, C, dan D untuk setiap mata kuliah mahasiswa. Diasumsikan sks setiap mata kuliah selalu 3. Kemudian bobot dari masing-masing nilai adalah: A=4, B=3, C=2, D=1.

Output program ialah hasil IPS yang didapatkan. Jalannya program seperti pada Gambar 5.7. Tips: Gunakan kontrol percabangan di dalam perulangan.



```
Command Prompt
Microsoft Windows [Version 10.0.18362.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Kuncoro Saputra>python "F:\Pyhton Workspace\Algoritma dan Pemrograman\modul4.py"
Program penghitung IPS Mahasiswa
Berapa jumlah mata kuliah? 6
Nilai MK 1: A
Nilai MK 2: B
Nilai MK 3: C
Nilai MK 4: A
Nilai MK 5: D
Nilai MK 6: C
Nilai IPS anda semester ini 2.67
```

Gambar 5.7: Hasil luaran program IPS mahasiswa

6. Percabangan dan Perulangan Kompleks

6.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan dan menggunakan struktur kontrol percabangan dan perulangan kompleks.
2. Dapat membuat program dengan struktur kontrol percabangan dan perulangan kompleks

6.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

6.3 Materi

6.3.1 Struktur Percabangan Kompleks

Percabangan di mana kondisi pemilihan tidak hanya satu tetapi bisa terdiri atas banyak alternatif. Perintah-perintah yang dikerjakan juga bisa lebih dari satu. Percabangan kompleks bentuk 1:

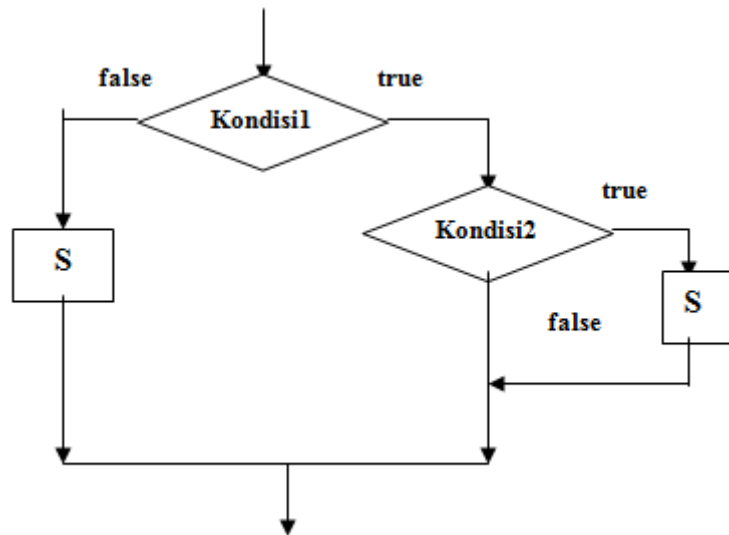
```
1 if kondisi1:
2     if kondisi2:
```

```

3         S
4         S
5     else:
6         S
7         S

```

Gambar flowchart bentuk 1 ada pada 6.1.



Gambar 6.1: Flowchart Percabangan Kompleks Bentuk 1

Percabangan kompleks bentuk 2:

```

1  if kondisi1:
2      if kondisi2:
3          S
4          S
5      else:
6          S
7          S
8  else:
9      S
10     S

```

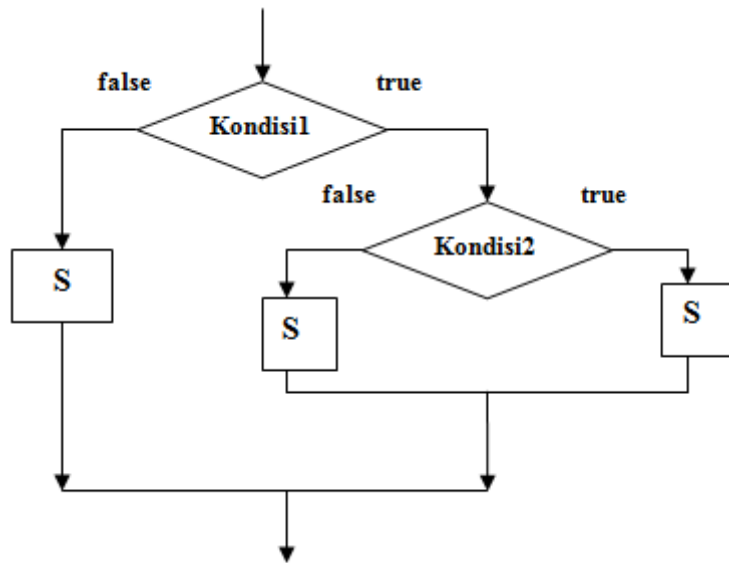
Gambar flowchart bentuk 2 ada pada 6.2

Percabangan kompleks bentuk 3:

```

1  if kondisi1:
2      S
3      if kondisi2:
4          S
5          S
6  else:
7      if kondisi3:

```



Gambar 6.2: Flowchart Percabangan Kompleks Bentuk 2

```

8         S
9         S
10      else:
11         S
12         S

```

Gambar flowchart bentuk 3 ada pada 6.3.

Percabangan kompleks bentuk 4:

```

1  if kondisi1:
2      S
3      if kondisi2:
4          S
5          S
6      else:
7          S
8          S
9  else:
10     if kondisi3:
11         S
12         S
13     else:
14         S
15         S
16 S

```

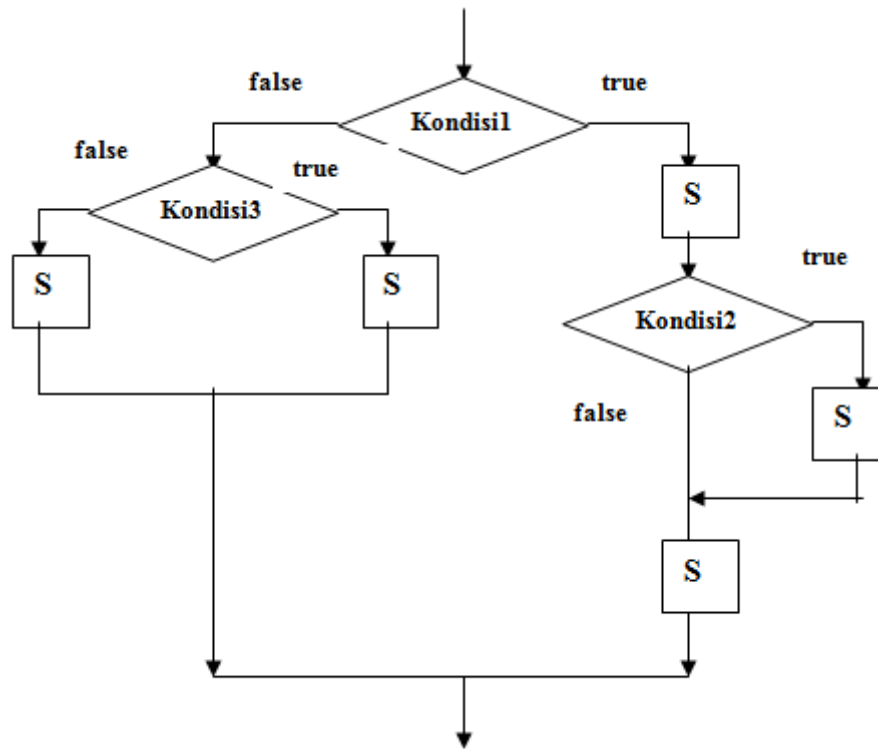
Gambar flowchart bentuk 4 ada pada 6.4.

Percabangan kompleks bentuk 5:

```

1  if kondisi1:
2      if kondisi2:

```



Gambar 6.3: Flowchart Percabangan Kompleks Bentuk 3

```

3         if kondisi3:
4             if kondisi4:
5                 S

```

Gambar flowchart bentuk 5 ada pada 6.5.
Percabangan kompleks bentuk 6:

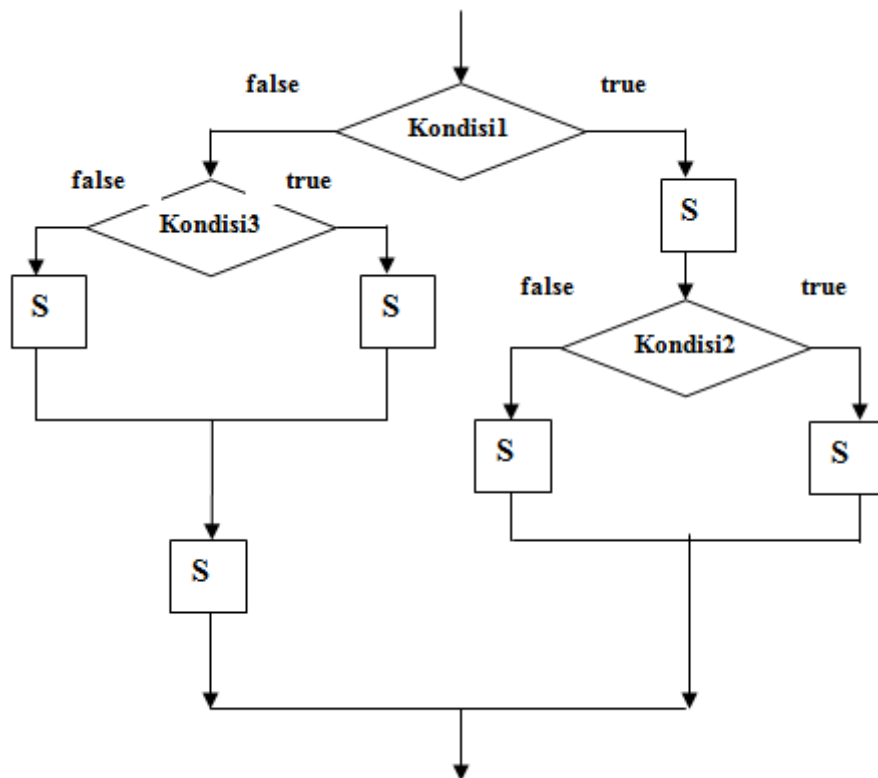
```

1  if kondisi1:
2      S
3  else:
4      if kondisi2:
5          S
6      else:
7          if kondisi3:
8              S
9          else:
10             if kondisi4:
11                 S
12             else:
13                 S

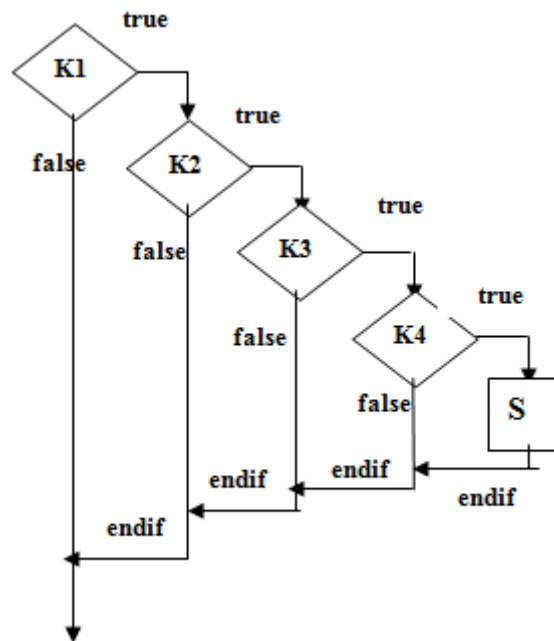
```

Gambar flowchart bentuk 6 ada pada 6.6.

Pada kasus tertentu IF bertingkat memiliki keuntungan tersendiri. Dengan menggunakan IF bertingkat maka eksekusi perintah menjadi lebih baik dan efisien. Dengan menggunakan IF bertingkat maka tidak semua kondisi IF dikerjakan sehingga waktu eksekusi lebih cepat. Sedangkan

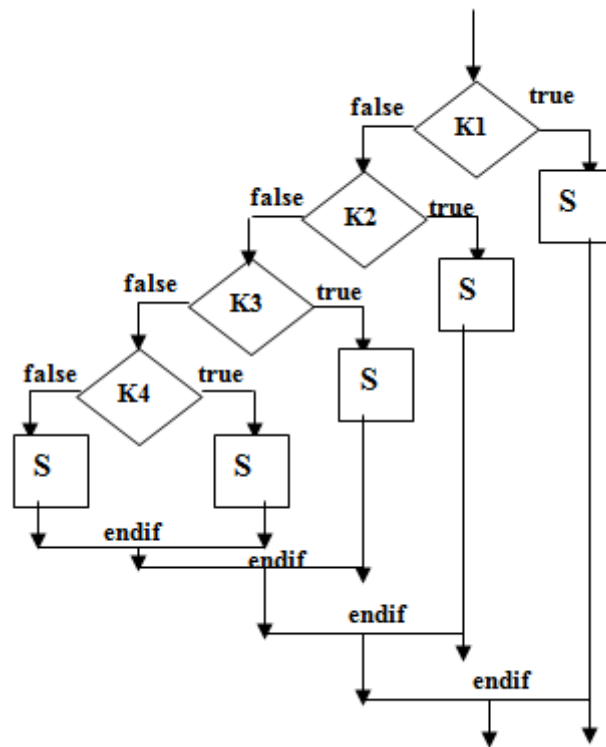


Gambar 6.4: Flowchart Percabangan Kompleks Bentuk 4



Gambar 6.5: Flowchart Percabangan Kompleks Bentuk 5

pada IF biasa semua kondisi IF pasti akan dicoba satu per satu walaupun mungkin pada akhirnya hanya satu saja perintah IF yang terpenuhi. Hal ini tentu memperlambat proses. Contoh kasus yang cocok untuk IF bertingkat adalah kasus konversi nilai angka menjadi nilai huruf dengan



Gambar 6.6: Flowchart Percabangan Kompleks Bentuk 6

batasan-batasan nilai yang sudah ditentukan sebelumnya.

```

1  if (kondisi1):
2      instruksi1
3  elif(kondisi2):
4      instruksi2
5  elif(kondisi3):
6      instruksi3
7  elif(kondisi4):
8      instruksi4
  
```

Bedakan dengan:

```

1  if (kondisi1):
2      instruksi1
3  if(kondisi2):
4      instruksi2
5  if(kondisi3):
6      instruksi3
7  if(kondisi4):
8      instruksi4
  
```

6.3.2 Struktur Perulangan Kompleks

Break

Perintah ini digunakan untuk menghentikan proses perulangan yang sedang terjadi. Biasanya disebabkan oleh suatu kondisi tertentu yang diimple-mentasikan menggunakan perintah IF.

```
1 for i in range(1000):  
2     print(i)  
3     if i==10:  
4         break
```

Output:

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Program di atas akan menampilkan angka 1 sampai dengan 10 walaupun di perulangan sudah diset dari 1 sampai dengan 1000. Hal ini karena perintah break yang diberikan pada saat kondisi i=10. Angka 10 masih ditampilkan karena perintah untuk mencetak diletakkan sebelum perintah break.

Gambar break dari kode program demo break ada pada 6.7:

Contoh program break lain:

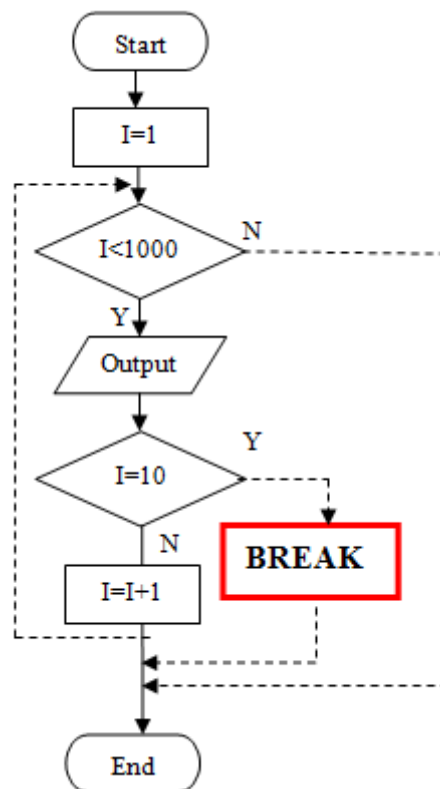
```
1     for i in range(1000):  
2         if i==10:  
3             break  
4         print(i)
```

Output:

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Program di atas akan menampilkan angka 1 sampai dengan 9 walaupun di perulangan sudah diset dari 1 sampai dengan 1000. Hal ini karena perintah break yang diberikan pada saat kondisi i=10. Angka 10 tidak ditampilkan karena perintah untuk mencetak diletakkan sesudah perintah break.

Gambar break dari kode program demo break ada pada 6.8:



Gambar 6.7: Flowchart Program Menggunakan Break

Continue

Perintah continue menyebabkan proses perulangan kembali ke awal mula, dengan mengabaikan statement-statement berikutnya setelah continue. Biasanya perintah continue juga diimplementasikan menggunakan perintah IF.

Contoh program continue:

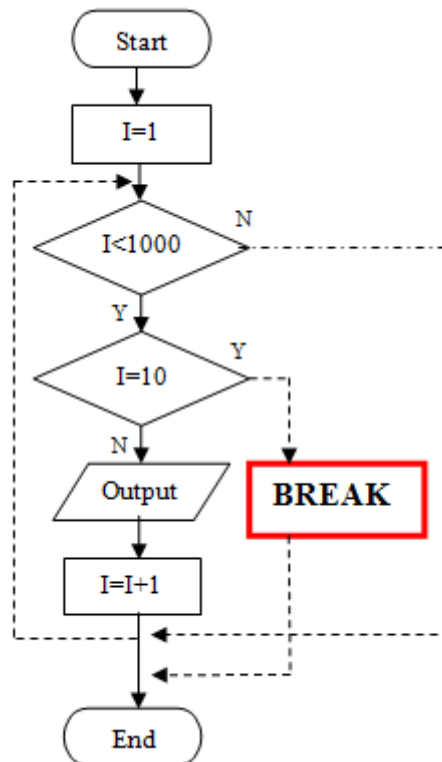
```

1   for i in range(10):
2       if i==5:
3           continue
4       print(i)
  
```

Output:

0
1
2
3
4
6
7
8
9

Program di atas tidak menampilkan angka 5 karena pada saat angka 5 akan ditampilkan, perintah



Gambar 6.8: Flowchart Program Menggunakan Break 2

continue dijalankan, sehingga perintah mencetak di bagian bawahnya tidak akan dikerjakan dan langsung dilanjutkan ke perulangan berikutnya!

Gambar break dari kode program demo continue ada pada 6.9:

Perulangan Bertingkat

Yang dimaksud dengan struktur perulangan kompleks adalah bentuk per-ulangan di mana di dalam suatu perulangan terdapat perulangan lain, sehingga terjadi perulangan bertingkat yang mengakibatkan waktu proses semakin lama. Ada banyak algoritma tertentu yang tepat untuk meng-gunakan struktur perulangan kompleks. Masalah yang dapat diselesaikan dengan perulangan kompleks adalah masalah matriks yang menggunakan array 2 dimensi, masalah game board seperti catur dan minesweeper, ataupun masalah pengolahan citra digital seperti algoritma untuk mendeteksi tepi citra, algoritma untuk mengubah citra berwarna menjadi grayscale, atau berbagai hal lain. Intinya, masalah yang dapat diselesaikan menggunakan perulangan kompleks biasanya memiliki pola grid (kotak-kotak) yang memiliki lebar dan panjang. Berikut ini adalah sebuah contoh perulangan kompleks:

Program 1:

```

1 for i in range(m):
2     <lakukan perintah ini>
3     <lakukan perintah itu>

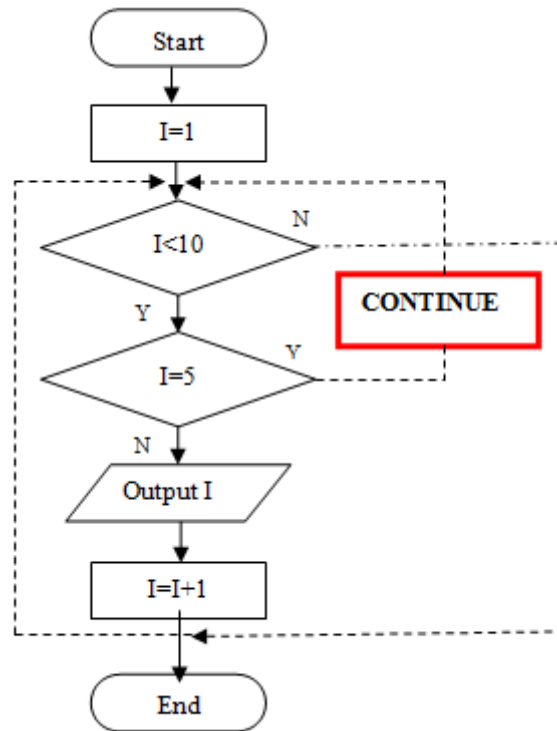
```

Program 2:

```

1 for j in range(n):

```



Gambar 6.9: Flowchart Program Menggunakan Continue

```

2     <lakukan perintah ini>
3     <lakukan perintah itu>

```

Program 1 mengerjakan perulangan sebanyak 10 kali, sedangkan program 2 mengerjakan perulangan sebanyak 5 kali. Kedua program tersebut berjalan sendiri-sendiri. Jika perulangan pada program 2 "dimasukkan" ke dalam perulangan program 1, maka menjadi:

```

1  for i in range(m):
2      for j in range(n):
3          <lakukan perintah ini di inner>
4          <lakukan perintah itu di inner>
5      <lakukan perintah lain di outer>
6      <lakukan perintah lain lagi di outer>

```

Bagian for i menjadi outer loop, sedangkan bagian for j menjadi inner loop. Alur pengerjaannya adalah sebagai berikut: untuk setiap 1x s/d m outer loop dijalankan akan dikerjakan n kali inner loop.

Untuk sintaks dalam bentuk sama saja. Perhatikan contoh berikut:

```

1  while(i<=m):
2      while(j<=n):
3          <lakukan perintah ini di inner>
4          <lakukan perintah itu di inner>
5      <lakukan perintah lain di outer>
6      <lakukan perintah lain lagi di outer>

```

Bagian while i menjadi outer loop, sedangkan bagian while j menjadi inner loop. Alur pengerjaannya adalah sebagai berikut: untuk setiap 1x s/d m outer loop dijalankan akan dikerjakan n kali inner loop.

6.4 Kegiatan Praktikum

Praktikum akan membuat beberapa program menggunakan perulangan dan percabangan kompleks dengan beberapa studi kasus tampilan deret berikut:

■ **Contoh 6.1** Anda diminta untuk membuat suatu deret dengan tampilan sebagai berikut:

```
n=5
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Pembahasan: Untuk dapat membuat program seperti di atas logika adalah sebagai berikut:

- Karena n = misalnya 5 maka n digunakan sebagai jumlah baris, yang dalam perulangan nested adalah outer loop, i=1 s/d n.
- Untuk inner loop kita perhatikan bahwa untuk baris i=1 maka dilakukan perulangan ke kanan 1x (atau sama dengan i x)
- Untuk setiap loop pada inner, dilakukan j=1 s/d i, dan tampilkan i nya. Untuk inner loop jangan dilakukan enter, sehingga harus diperhatikan end=" pada printnya.
- Lakukan hal tersebut sampai selesai. Untuk setiap akhir j, lakukan enter (ganti baris)

Kode program dapat dibuat sebagai berikut:

```
1 n=int(input("Masukkan n = "))
2 for i in range(1,n+1):
3     for j in range(1,i+1):
4         print(i," ",end='')
5     print()
```

■
■ **Contoh 6.2** Anda diminta untuk membuat suatu deret dengan tampilan sebagai berikut:

```
n=5
1 2 3 4 5
5 4 3 2 1
1 2 3 4 5
5 4 3 2 1
1 2 3 4 5
```

Pembahasan: Untuk dapat membuat program seperti di atas logika adalah sebagai berikut:

- Karena n = misalnya 5, maka n akan menjadi outer loop (baris). i=1 s/d n.
- Untuk inner loop kita juga akan melakukan perulangan smp dengan n atau turun dari n s/d 1, tergantung barisnya genap atau ganjil. Jika ganjil inner loop bergerak naik, jika genap maka inner loop bergerak turun, j=n s/d 0, step -1.
- Untuk setiap loop pada inner, setelah habis maka lakukan enter / ganti baris dari outer loop.

Kode program dapat dibuat sebagai berikut:

```

1         n=int(input("Masukkan n = "))
2         for i in range(1,n+1):
3             if i%2==1:
4                 for j in range(1,n+1):
5                     print(j," ",end='')
6             else:
7                 for j in range(n,0,-1):
8                     print(j," ",end='')
9         print()

```

■ **Contoh 6.3** Anda diminta untuk membuat suatu deret dengan tampilan sebagai berikut:

```

n=6
X O X O X O
X O X O X
X O X O
X O X
X O
X

```

Pembahasan: Untuk dapat membuat program seperti di atas logika adalah sebagai berikut:

- Karena n = misalnya 6, maka n akan menjadi outer loop (baris). i=0 s/d n.
- Untuk inner loop kita juga akan melakukan perulangan dari 1 smp dengan n-i. Untuk setiap j ganjil maka tampilkan 'X', sedangkan jika genap maka tampilkan 'O'.
- Untuk setiap loop pada inner, setelah habis maka lakukan enter / ganti baris dari outer loop.

Kode program dapat dibuat sebagai berikut:

```

1         n=int(input("Masukkan n = "))
2         for i in range(0,n+1):
3             for j in range(1,n-i+1):
4                 print("X",end='') if j%2==1 else print("O",end='')
5         print()

```

■ **Contoh 6.4** Anda diminta untuk membuat suatu deret dengan tampilan pada gambar 6.10.

Pembahasan: Untuk dapat membuat kode program seperti di atas, maka logikanya adalah sebagai berikut:

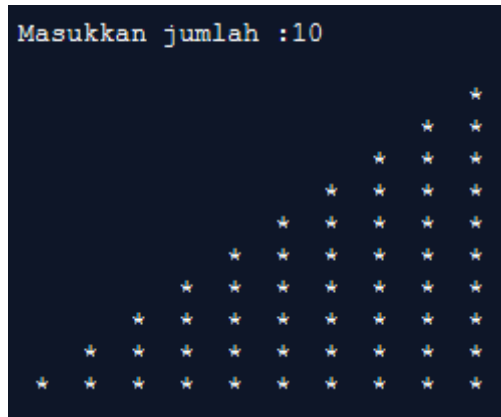
- Karena n = misalnya 10, maka n akan menjadi outer loop (baris). i=n turun smp dengan 1.
- Untuk inner loop kita harus menuliskan terlebih dahulu spasi kosong sebanyak n-i-1. Setelah itu digabungkan dengan menampilkan bintang (*) sebanyak sisanya (1 s/d n-i-1)
- Untuk setiap loop pada inner, setelah habis maka lakukan enter / ganti baris dari outer loop.

Kode program dapat dibuat sebagai berikut:

```

1         hasil = ""
2
3         x = int(input("Masukkan jumlah :"))
4         bar = x

```



Gambar 6.10: Soal Segitiga

```

5      # Looping Baris
6      while bar >= 0:
7
8          # Looping Kolom Spasi Kosong
9          kol = bar
10         while kol > 0:
11             hasil += "  "
12             kol -= 1
13
14         # Looping Kolom Bintang
15         kanan = 1
16         while kanan < (x - (bar-1)):
17             hasil += " * "
18             kanan += 1
19
20         hasil = hasil + "\n"
21         bar -= 1
22
23     print (hasil)

```

■

6.5 Latihan Mandiri

Latihan 6.1 Buatlah program untuk mencari bilangan prima terdekat dari suatu bilangan yang diinputkan oleh pengguna (n) dan nilai bilangan prima tersebut < n. Contoh: input n=12, maka prima terdekat < 12 adalah 11 Contoh: input n=21, maka prima terdekat < 21 adalah 19 ■

Latihan 6.2 Buatlah program untuk menampilkan deret seperti di bawah ini. n diinputkan secara dinamis

```
contoh: n = 6
720 6 5 4 3 2 1
120 5 4 3 2 1
24 4 3 2 1
6 3 2 1
2 2 1
1 1
```

Latihan 6.3 Buatlah program untuk menampilkan deret seperti di bawah ini. n diinputkan secara dinamis

```
contoh: tinggi = 5, lebar = 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
```



String dan File

7	Pengolahan String	83
7.1	Tujuan Praktikum	
7.2	Alat dan Bahan	
7.3	Materi	
7.3.1	Pengantar String	
7.3.2	Pengaksesan String dan Manipulasi String	
7.3.3	Operator dan Metode String	
7.3.4	Parsing String	
7.4	Kegiatan Praktikum	
7.4.1	Pembahasan Soal 1	
7.4.2	Pembahasan Soal 2	
7.4.3	Pembahasan Soal 3	
7.4.4	Pembahasan Soal 4	
7.5	Latihan Mandiri	
8	Membaca dan Menulis File	91
8.1	Tujuan Praktikum	
8.2	Alat dan Bahan	
8.3	Materi	
8.3.1	Pengantar File	
8.3.2	Pengaksesan File	
8.3.3	Manipulasi File	
8.3.4	Penyimpanan File	
8.4	Kegiatan Praktikum	
8.5	Latihan Mandiri	

7. Pengolahan String

7.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan tentang String.
2. Dapat mengakses String dan memanipulasi String.
3. Dapat menggunakan operator metode String.
4. Dapat melakukan parsing String sederhana.

7.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

7.3 Materi

7.3.1 Pengantar String

String adalah untaian karakter-karakter yang menjadi satu kesatuan dan digunakan dalam program komputer untuk menyimpan kalimat, baik panjang ataupun pendek. String adalah suatu jenis data yang mampu menyimpan huruf / karakter dan disimpan dalam kode ASCII. Tidak semua bahasa pemrograman memiliki tipe data String, seperti misalnya bahasa C. Tipe data ini merupakan

jenis tipe data yang bukan tipe data dasar, karena tipe data ini pada dasarnya menyimpan lebih dari satu nilai tunggal sebagai satu kesatuan. Beberapa bahasa pemrograman ada yang menyebutkan bahwa String pada dasarnya adalah kumpulan tipe data karakter / array of character / list of character.

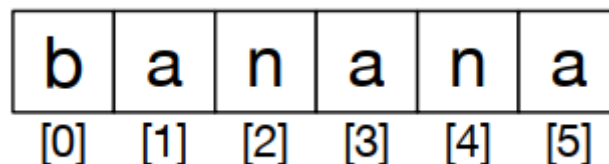
7.3.2 Pengaksesan String dan Manipulasi String

String dapat dibuat dengan sederhana menggunakan variabel:

```
1  namasaya = "Antonius Rachmat C"
2  temansaya1 = "Yuan Lukito"
3  temansaya2 = 'Laurentius Kuncoro'
4  temansaya3 = "Matahari" + 'Bakti'
5
6  print(temansaya3)
7  print(namasaya[0]) #'A'
8  print(namasaya[9]) #'R'
9  print(temansaya1[1]) #'u'
10
11  huruf = temansaya2[0]
12  print(huruf) #'L'
```

Jadi string pertama kali dibuat dengan deklarasi variabel dan langsung diisi dengan data, string dapat diakses sebagai satu kesatuan dengan menyebut nama variabelnya, atau per huruf dengan menyebutkan indeksinya. Indeks pada string dimulai dari 0 seperti layaknya pada list. Indeks string haruslah berupa bilangan bulat, bukan pecahan.

Pada memory komputer, string disimpan secara urut menggunakan list yang berisi huruf-huruf dengan indeks yang dimulai dari nol. Gambar 7.1 menunjukkan string berisi "banana" yang disimpan pada memory komputer.



Gambar 7.1: Bentuk string di dalam memory

7.3.3 Operator dan Metode String

OPERATOR in

Pada String kita dapat memeriksa apakah suatu kalimat merupakan substring dari suatu kalimat lain dengan menggunakan operator in. Hasil dari operator ini adalah True / False.

```
1  kalimat = "saya mau makan"
2  data = "saya"
3  print(data in kalimat) #True
4  print("mau" in kalimat) #True
5  print("dia" in kalimat) #False
```

Selain operator in, pada String juga dapat dilakukan perbandingan (comparison) yang juga menghasilkan True atau False.

```
1     if "saya" > "dia":
2         print("Ya") #Ya
3     else:
4         print("Tidak")
5
6     if "dua" == "dua":
7         print("Sama")    #Sama
```

FUNGSI len

Cara untuk mengetahui berapa panjang (berapa jumlah karakter) dari sebuah string adalah dengan menggunakan operator len(<string>). Untuk menampilkan huruf terakhir dari sebuah string kita harus menggunakan indeks string yang ke- len(<string>-1), sebab indeks dimulai dari 0.

Contoh program Python adalah:

```
1     kalimat = "universitas kristen duta wacana yogyakarta"
2     print(len(kalimat)) #output 42
3
4     terakhir = kalimat[len(kalimat)-1]
5     print(terakhir) #output 'a'
6
7     #bisa juga menggunakan indeks -1
8     terakhir_versi2 = kalimat[-1]
9     print(terakhir_versi2) #output 'a'
10    #atau menggunakan indeks -2 untuk huruf terakhir kedua
11    terakhir2 = kalimat[-2]
12    print(terakhir2)    #output 't'
```

TRAVERSING STRING

Untuk dapat menampilkan string dengan cara ditampilkan huruf demi huruf adalah dengan menggunakan loop yang dilakukan per huruf dengan 2 cara:

- Dilakukan dengan akses terhadap indeks

```
1     kalimat = "indonesia jaya"
2     i = 0
3     while i < len(kalimat):
4         print(kalimat[i],end=' ')
5         i += 1
```

- Dilakukan tanpa akses terhadap indeks secara otomatis

```
1     kalimat = "indonesia jaya"
2     for kal in kalimat:
3         print(kal,end=' ')
```

STRING SLICE

String slice adalah menampilkan substring pada sebuah string dengan menggunakan indeks dari awal tertentu sampai akhir-1 tertentu. Sintaksnya menggunakan `<string>[awal:akhir]`. Bagian awal atau akhir boleh dikosongkan. Bagian awal dimulai dari 0.

```
1 kalimat = "cerita rakyat"
2 awal = 0
3 akhir = 6
4 print(kalimat[awal:akhir]) #cerita
5 print(kalimat[7:len(kalimat)]) #rakyat
6 print(kalimat[:5]) #cerit
7 print(kalimat[5:]) #a rakyat
8 print(kalimat[:]) #cerita rakyat
```

String merupakan data yang bersifat immutable! Immutable adalah bahwa data tersebut tidak bisa diubah saat program berjalan, hanya bisa diinisialisasi saja. Contoh:

```
1 kalimat = "satu"
2 kalimat[0] = "batu" #TypeError: 'str' object does not support item assignment
```

Agar bisa diubah, maka harus disimpan dalam variabel yang berbeda.

```
1 kalimat = "satu"
2 kalimat_baru = kalimat[0] + "alah" #salah
```

Berikut adalah beberapa method String yang sering digunakan:

Nama Method	Kegunaan	Penggunaan
<code>capitalize()</code>	untuk mengubah string menjadi huruf besar	<code>string.capitalize()</code>
<code>count()</code>	menghitung jumlah substring yang muncul dari sebuah string	<code>string.count()</code>
<code>endswith()</code>	mengetahui apakah suatu string diakhiri dengan string yang diinputkan	<code>string.endswith()</code>
<code>startswith()</code>	mengetahui apakah suatu string diawali dengan string yang diinputkan	<code>string.startswith()</code>
<code>find()</code>	mengembalikan indeks pertama string jika ditemukan string yang dicari	<code>string.find()</code>
<code>islower()</code> dan <code>isupper()</code>	mengembalikan True jika string adalah huruf kecil / huruf besar	<code>string.islower()</code> dan <code>string.isupper()</code>
<code>isdigit()</code>	mengembalikan True jika string adalah digit (angka)	<code>string.isdigit()</code>
<code>strip()</code>	menghapus semua whitespace yang ada di depan dan di akhir string	<code>string.strip()</code>
<code>split()</code>	memecah string menjadi token-token berdasarkan pemisah, misalnya berdasarkan spasi	<code>string.split()</code>

Dan masih banyak lainnya. Ingat semua fungsi/method di atas yang mengembalikan string, mengembalikan string baru, tidak mengubah yang aslinya, karena string bersifat *immutable*. Daftar lengkap dapat dilihat di <https://docs.python.org/library/stdtypes.html#string-methods>

Operator * dan + pada String

Pada Python operator + dan * memiliki kemampuan khusus. Operator + yang biasanya digunakan untuk menjumlahkan bilangan bisa digunakan untuk menggabungkan dua buah string. Sedangkan operator * yang bisa digunakan untuk mengkalikan bilangan bisa digunakan untuk menampilkan string sejumlah perkaliannya. Perhatikan kode berikut:

```
1 kata1 = "saya"
2 kata2 = "makan"
3 kata3 = kata1 + " " + kata2
4 print(kata3)      #hasil adalah penggabungan: saya makan
5 kata4 = "ulang"
6 print(kata4 * 4)   #hasil adalah ulangulangulangulang
7 kata4 = "ulang "
8 print(kata4 * 2)   #hasil adalah ulang ulang
```

7.3.4 Parsing String

Parsing string adalah cara menelusuri string bagian demi bagian untuk mendapatkan / menemukan / mengubah bagian string yang diinginkan. Perhatikan contoh berikut:

"Saudara-saudara, pada tanggal 17-08-1945 Indonesia merdeka".

Dilihat dari string di atas, diinginkan untuk mengambil tanggal bulan tahun dari kalimat di atas dan disusun ulang menjadi format 08/17/1945. Kita dapat menggunakan cara parsing string misalnya:

- Spit string berdasarkan spasi sehingga menjadi token: "Saudara-saudara", "pada", "tanggal", "17-08-1945", "Indonesia", dan "merdeka".
- Dari sini, lanjutkan dengan mencari token yang diawali dengan angka, kemudian lanjutkan dengan split angka tersebut dengan pemisah '-'.
- Kemudian susun ulang token-token dari langkah sebelumnya sehingga berbentuk seperti yang diinginkan.

Berikut adalah contoh penyelesaiannya:

```
1 kalimat = "Saudara-saudara, pada tanggal 17-08-1945 Indonesia merdeka"
2
3 hasil = kalimat.split(" ")
4
5 for kal in hasil:
6     if kal[0].isdigit():
7         hasil2 = kal.split("-")
8         print(hasil2[1]+"/"+hasil2[0]+"/"+hasil2[2])
```

Hasil dari keluaran program adalah **08/17/1945**

7.4 Kegiatan Praktikum

Kasus 7.1 Buatlah program untuk menghitung berapa huruf hidup pada sebuah kalimat per kata

Kasus 7.2 Buatlah program untuk membuat mengubah susunan format tanggal dari YYYY-MM-DD menjadi DD-MM-YYYY

Kasus 7.3 Buatlah program untuk mengetahui apakah suatu kalimat adalah palindrom atau bukan! Palindrom adalah kalimat yang jika dibalik sama saja. Misalnya: Step on no pets, Pull up If I pull up, Some men interpret nine memos, dan Madam, In Eden I'm Adam.

Kasus 7.4 Buatlah program untuk mengambil beberapa kata dari suatu kalimat yang diinputkan. Misal kalimat: "saya akan pergi sekolah". Akan diambil 1 kata, maka hasilnya: "saya", "akan", "pergi", "sekolah". Sedangkan jika 2 kata maka hasilnya: "saya akan", "akan pergi", "pergi sekolah".

7.4.1 Pembahasan Soal 1

Untuk menghitung huruf hidup digunakan tahapan sebagai berikut:

- minta input kalimat yang akan dicari
- ubah string menjadi huruf kecil semua agar tidak membedakan huruf besar dan kecil
- siapkan variabel total jumlah huruf hidup aiueo dan set nilai awal = 0
- carilah ada berapa huruf 'a','i','u','e','o' pada kalimat yang diinputkan dengan method count()

Hasil:

```
1 a_string = "AnTonIus"
2 lowercase = a_string.lower()
3 total = 0
4 for x in "aiueo":
5     jml = lowercase.count(x)
6     total += jml
7 print(total)      #hasil = 4
```

7.4.2 Pembahasan Soal 2

Langkah yang harus dilakukan bisa 2 macam, yaitu dengan memecah / split string dan menyusunnya kembali sesuai yang diinginkan.

```
1 tgl = "2020-12-01"
2 hasil = tgl.split("-")
3 tgl2 = hasil[2]+"-"+hasil[1]+"-"+hasil[0]
4 print(tgl2)
```

7.4.3 Pembahasan Soal 3

Palindrom dapat diperiksa dengan membalik kata tersebut dan kemudian mencocokkannya dengan string aslinya. Jika sama maka palindrom, jika tidak maka bukan palindrom.

```
1 satu = "Step! on, no.. pets??"
2 satu = satu.lower()
3
4 satu = ''.join([i for i in satu if i.isalpha()])      #buang semua yang bukan alfabet
5
6 dua = satu[::-1]
7 if dua == satu:
8     print("palindrom")
9 else:
10    print("bukan")
```

7.4.4 Pembahasan Soal 4

Untuk mengambil beberapa kata dari kalimat, digunakan logika sebagai berikut:

- Ubah kalimat menjadi huruf kecil semua (lowercase)
- Pecah (split), kalimat dipisah berdasarkan spasi
- Untuk setiap kata yang sudah dipisah tersebut, maka ambil indeks kata per n buah, misal 1 buah jika n=1, 2 jika n=2. Indeks yang diambil adalah dimulai dari 0, jika n=2 maka ambil 0 dan 1, lalu ambil 1 dan 2, 2 dan 3, 3 dan 4, dst...dilakukan dalam perulangan.

```
1  #Kalimat
2  text = 'A quick brown fox jumps over the lazy dog.'
3
4  def ambil_kata_kalimat(kalimat, n):
5      kalimat = kalimat.lower()    #ubah huruf kecil
6      hasil_akhir = []             #siapkan tempat hasil
7      hasil = kalimat.split()      #pecah berdasarkan spasi
8      for i in range(0, len(hasil)): #loop per hasil pecah
9          tmp = ' '.join(hasil[i:i + n]) #ambil per indeks
10         hasil_akhir.append(tmp) #tambahkan ke list
11
12     return hasil_akhir            #return
13
14 print(ambil_kata_kalimat(text, 2))
```

Hasil

```
['a quick', 'quick brown', 'brown fox', 'fox jumps',
'jumps over', 'over the', 'the lazy',
'lazy dog.', 'dog.']
```

7.5 Latihan Mandiri

Latihan 7.1 Buatlah sebuah program yang dapat mendeteksi apakah suatu kata adalah anagram dari kata lainnya atau bukan! Anagram adalah kata yang dibolak-balik susunan hurufnya sama. Misal: mata anagram dengan atma, maat, taam, tama, dsb. ■

Latihan 7.2 Buatlah suatu program yang dapat menghitung frekuensi kemunculan suatu kata yang ada pada String. Misal terdapat kalimat "Saya mau makan. Makan itu wajib. Mau siang atau malam saya wajib makan". Ditanyakan kata "makan". Output: makan ada 3 buah ■

Latihan 7.3 Buatlah suatu program yang dapat menghapus semua spasi yang berlebih pada sebuah string, dan menjadikannya satu spasi normal! Misal: "saya tidak suka memancing ikan " Output: "saya tidak suka memancing ikan" ■

Latihan 7.4 Buatlah suatu program mengetahui kata terpendek dan terpanjang dari suatu kalimat yang diinputkan! Misal: "red snakes and a black frog in the pool" Output: terpendek: a, terpanjang: snakes ■

8. Membaca dan Menulis File

8.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan tentang File.
2. Dapat mengakses File dan memanipulasinya.
3. Dapat menggunakan File untuk penyimpanan data program.

8.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.
3. File mbox.txt dan mbox-short.txt (ada di eclass)

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

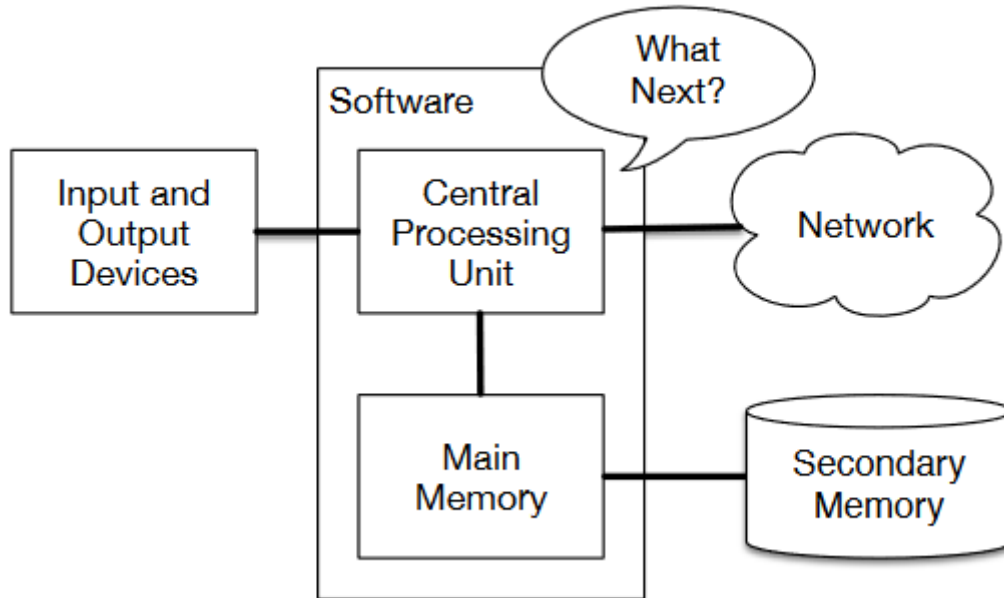
8.3 Materi

8.3.1 Pengantar File

Program yang berjalan membutuhkan memory primer di dalam komputer. Semua data yang ada di program tersebut disimpan di dalam memory dan ketika program selesai dijalankan dan dimatikan, maka semua data di dalam program tersebut juga ikut hilang. Penyimpanan data di dalam

memory bersifat tidak permanen (volatile). Karena sifat tersebut, program yang menggunakan memory primer tidak akan dapat menyimpan data setelah program dimatikan.

Untuk bisa menyimpan data pada program harus digunakan penyimpanan tetap yaitu secondary memory. Secondary memory dapat dilihat pada gambar 8.1



Gambar 8.1: Secondary Memory di Komputer

File disimpan pada secondary memory sehingga file dapat digunakan untuk menyimpan data dari program dan tidak akan hilang walaupun komputer dimatikan. File pada dasarnya adalah bit-bit data yang disimpan di dalam secondary memory secara permanen, berupa kumpulan informasi yang saling berelasi satu sama lain sebagai satu kesatuan. File bisa berupa file system, file program (binary), file multimedia, file teks, dan lain sebagainya. File memiliki property seperti nama file, ukuran, letak di harddisk, owner, hak akses, tanggal akses, dan lain-lain. Contoh property sebuah file dapat dilihat dari gambar 8.2

8.3.2 Pengaksesan File

Untuk dapat mengakses file, langkah-langkah yang harus dilakukan adalah:

1. Menyiapkan file dan path yang akan diakses
2. Open file
3. Lakukan sesuatu dengan file tersebut, seperti ditampilkan (read) isinya atau diubah / ditulisi (write)
4. Close file

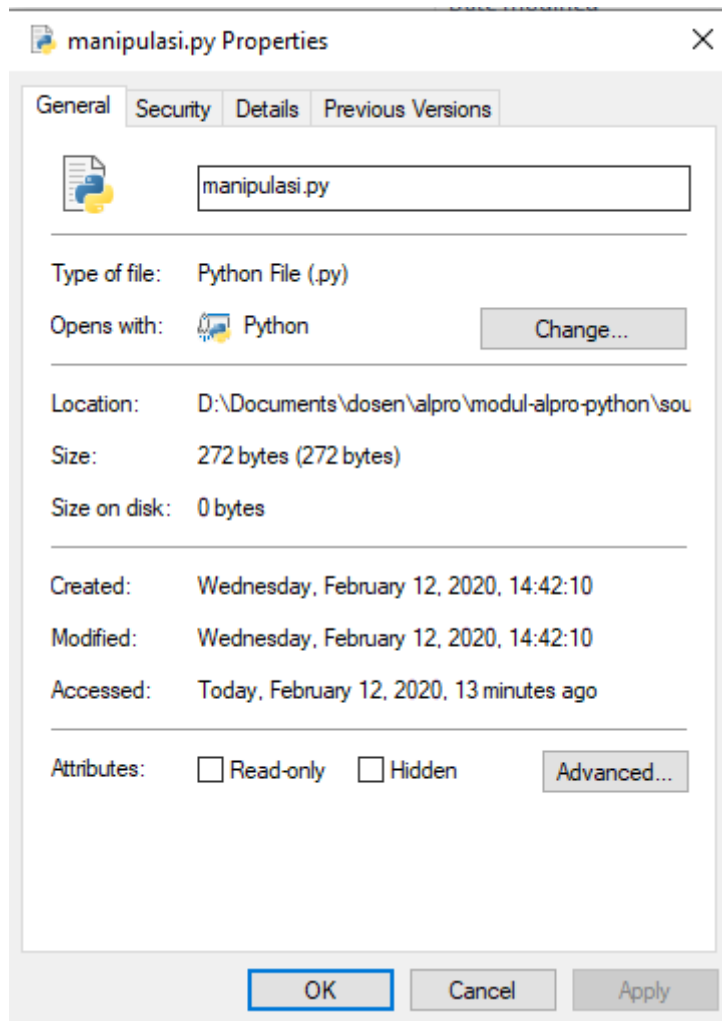
Gambaran handle file dapat dilihat pada 8.3

Program Python dapat dibuat sebagai berikut:

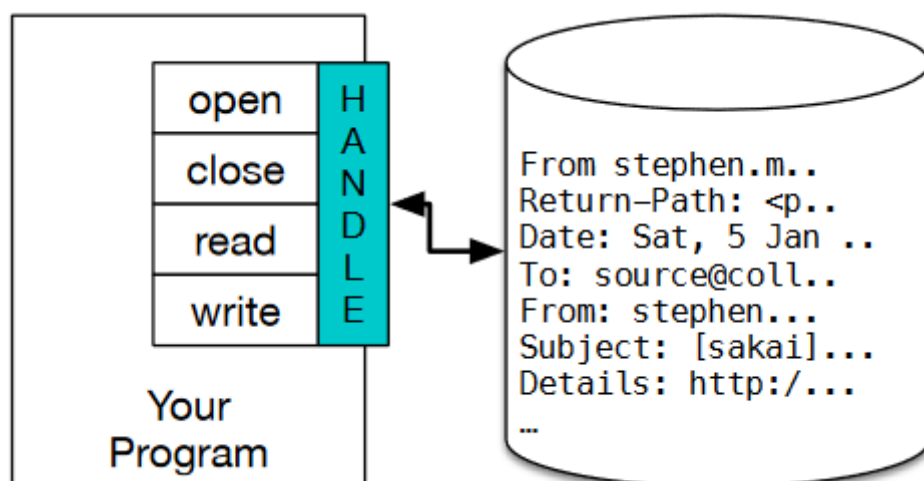
```
1 handle = open('mbox-short.txt')
2 print(handle)
```

Hasil adalah:

```
1 <_io.TextIOWrapper name='mbox-short.txt' mode='r' encoding='UTF-8'>
```



Gambar 8.2: Contoh Property File



Gambar 8.3: Ilustrasi Handle File

Jadi hasilnya berupa tampilan nama file, modenya (r = read), dan encoding yang digunakan yaitu unicode UTF-8 dari sistem io pada Python. Jika nama file tidak ada / tidak ditemukan, maka output akan error:

```
1 Traceback (most recent call last): File "main.py", line 1, in <module>
2 handle = open('tidak-ada.txt')
3 FileNotFoundError: [Errno 2] No such file or directory: 'tidak-ada.txt'
```

Pada file teks, biasanya file akan terdiri dari baris demi baris string. Cara pembacaan file teks biasanya juga menggunakan model baca baris demi baris untuk setiap string yang ditemukan sampai dengan EOF (End of File)

Contoh tampilan baris-baris dari sebuah file teks mbox-short.txt adalah pada gambar 8.4

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
...
```

Gambar 8.4: Contoh File Teks dan Barisnya

8.3.3 Manipulasi File

Untuk bisa memanipulasi file maka harus dimulai dari membaca file tersebut terlebih dahulu. Cara membaca file pada Python adalah:

1. Siapkan file
2. Open file
3. Loop setiap baris pada file
4. Close file

Cara pembacaan file adalah:

```
1 handle = open('mbox-short.txt')
2 count = 0
3 for line in handle:
4     count = count + 1
5 print('Line Count:', count)
```

Hasilnya adalah Line Count: 1910




Mengapa harus dibaca baris-perbaris? Untuk mengatasi kemungkinan sistem membuka file yang besar sekali dalam satu waktu dan akhirnya hang / crash karena ukuran file yang sangat besar.

Cara menampilkan ukuran file teks dalam bytes, dapat digunakan fungsi len dari string yang ada pada file.

```
1     handle = open('mbox-short.txt')
2     hasil = handle.read()
3     print("Ukuran: " + len(hasil) + "bytes")
4     print("Huruf dari belakang sendiri mundur 16 huruf adalah: " + hasil[-16::1])
```

Program di atas akan membuka file mbox-short.txt, menampilkan ukuran berapa banyak huruf yang ada pada file tersebut (catatan: kalau dianggap 1 karakter = 1 byte, maka bisa disebut juga ukuran berapa banyak karakter = ukuran file tersebut dalam byte), dan terakhir menampilkan string dari huruf paling belakang maju 16 huruf kedepan.

 **Hati-hati!** Perintah read() pada file sangat boros memory, sehingga jika ukuran file begitu besar maka lebih baik tidak menggunakan read(), melainkan menggunakan teknik loop seperti pada contoh di atas sebelumnya.

Selama dilakukan looping kita juga dapat melakukan manipulasi terhadap file tersebut, seperti misalnya menangkap / menampilkan bagian dari string. Seperti pada contoh file mbox, saat looping kita dapat menampilkan hanya kalimat yang diawali dengan "tanggal" saja, yaitu "Date :". Perhatikan contoh program berikut:

```
1     handle = open('mbox-short.txt')
2     count = 1
3     for line in handle:
4         if line.startswith("Date:") and count <= 10:
5             count += 1
6             print(line)
```

Program di atas menghasilkan output 10 baris yang berupa tanggal saja.

Date: Sat, 5 Jan 2008 09:12:18 -0500

Date: 2008-01-05 09:12:07 -0500 (Sat, 05 Jan 2008)

Date: Fri, 4 Jan 2008 18:08:57 -0500

Date: 2008-01-04 18:08:50 -0500 (Fri, 04 Jan 2008)

Date: Fri, 4 Jan 2008 16:09:02 -0500

Date: 2008-01-04 16:09:01 -0500 (Fri, 04 Jan 2008)

Date: Fri, 4 Jan 2008 15:44:40 -0500

Date: 2008-01-04 15:44:39 -0500 (Fri, 04 Jan 2008)

Date: Fri, 4 Jan 2008 15:01:38 -0500

Date: 2008-01-04 15:01:37 -0500 (Fri, 04 Jan 2008)

Mengapa ada baris kosong disetiap baris di atas? hal ini karena dari file mbox sudah terdapat newline di setiap baris dan ditambah perintah print yang kita buat, sehingga terjadi double newline. Silahkan modifikasi agar enter tidak double! Kita bisa menggunakan perintah `rstrip` pada line atau menggunakan perintah `print` tanpa `newline`.

8.3.4 Penyimpanan File

Dalam Python cara untuk menulis ke file adalah sama dengan cara membuka (`open`) file pada sub bab sebelumnya, hanya perlu mengubah metodenya saja yang tadinya `r` menjadi `w` sebagai berikut: **`fout=open('output.txt','w')`** Untuk menuliskan isi string ke dalam file `output.txt` langsung saja digunakan perintah `write(<string>)` dan jangan lupa tutup file dengan `close()`

Contoh lengkap adalah sebagai berikut:

```
1 handle = open('output.txt','w')
2 tulisan = "teks ini akan dituliskan ke file\n"
3 handle.write(tulisan)
4 handle.close()
```

8.4 Kegiatan Praktikum

Kegiatan praktikum akan dilakukan untuk memanipulasi lebih dalam lagi file teks `mbox-short.txt` sebagai berikut:

Kasus 8.1 Program harus mampu menerima nama file teks tertentu (`mbox-short.txt` atau `mbox.txt`) dan kemudian tampilkanlah **semua baris** yang mengandung string web pada file tersebut yang menggunakan domain berakhiran `*.ac.uk` dan **berapa jumlahnya**.

Pembahasan Soal 1

Untuk dapat menerima input nama file digunakan perintah `input`, kemudian masukkan nama file. File tersebut sebaiknya berada dalam satu folder yang sama agar mudah. Untuk menampilkan baris-baris file yang mengandung string URL web yang mengandung domain berakhiran: `*.ac.uk` digunakan perintah `find()`. Untuk menampilkan jumlah baris digunakan counter untuk setiap baris yang ditemukan. Berikut adalah kode programnya.

```
1 filename = input("nama file: ")
2 handle = open(filename)
3 c = 0
4 for line in handle:
5     if line.find("ac.uk") != -1:
6         c += 1
7     print("Web domain 'ac.uk' ditemukan di \"" + line.strip() + "\"")
8 print("Jumlah: ",c)
```

Silahkan lihat hasil programnya!

Kasus 8.2 Program harus mampu menerima nama file teks tertentu (`mbox-short.txt` atau `mbox.txt`) dan kemudian tampilkanlah **berapa baris** string pada file yang diawali kata "Subject" dan ubahlah semua baris tersebut menjadi "capitalized each word".

Pembahasan Soal 2

Untuk dapat menerima input nama file digunakan perintah input, kemudian masukkan nama file. File tersebut sebaiknya berada dalam satu folder yang sama agar mudah. Untuk menampilkan baris-baris file yang diawali string 'Subject:' digunakan perintah startswith(). Untuk menampilkan jumlah baris digunakan counter untuk setiap baris yang ditemukan. Berikut adalah kode programnya.

```
1 filename = input("nama file: ")
2 handle = open(filename)
3 c = 0
4 for line in handle:
5     if line.startswith("Subject:"):
6         c += 1
7         line = line.strip().title()
8         print(line)
9 print("Jumlah: ", c)
```

Silahkan lihat hasil programnya!

Kasus 8.3 Program harus mampu menampilkan ukuran file dalam KB dari sebuah file teks dan menghandle error jika file yang diinputkan tidak ditemukan.

Pembahasan Soal 3

Untuk dapat menerima input nama file digunakan perintah input, kemudian masukkan nama file. File tersebut sebaiknya berada dalam satu folder yang sama agar mudah. Untuk menghitung ukuran file dapat digunakan perintah len untuk setiap baris string yang diperoleh dari pembacaan baris file. Ukuran file yang didapat adalah berupa byte, sehingga perlu dibagi 1000 yang akan menjadi dalam KB. Untuk menghandle error, gunakan blok try catch. Berikut adalah kode programnya.

```
1 filename = input("nama file: ")
2 try:
3     handle = open(filename)
4     total = 0
5     for line in handle:
6         total += len(line)
7         kb = total / 1000
8         print("Ukuran: " + str(mb) + " KB")
9 except:
10    print("File tidak ditemukan!")
```

Silahkan lihat hasil programnya!

8.5 Latihan Mandiri

Latihan 8.1 Buatlah sebuah program yang dapat membandingkan 2 buah file teks dan kemudian menampilkan perbedaan antar kedua teks per barisnya jika ada perbedaan! ■

Latihan 8.2 Buatlah sebuah program untuk menampilkan soal sederhana yang diambil dari file teks soal.txt yang memiliki format sebagai berikut:

```
1+1 = || 2
Bendera Indonesia? || Merah Putih
Kota gudeg adalah: || Yogyakarta
Komponen PC untuk penyimpanan file adalah... || harddisk
50 * 20 = || 1000
```

Dari soal tersebut tampilkan sbb:

```
nama file1: soal.txt
1+1 =
Jawab: 2
Jawaban benar!
Bendera Indonesia?
Jawab: merah putih
Jawaban benar!
Kota gudeg adalah:
Jawab: yogya
Jawaban salah!
Komponen PC untuk penyimpanan file adalah...
Jawab: HARDDISK
Jawaban benar!
```

IV

Type Data Collection

9	Type Data List	101
9.1	Tujuan Praktikum	
9.2	Alat dan Bahan	
9.3	Materi	
9.3.1	Sifat-sifat List	
9.3.2	Mengakses dan Mengubah isi List	
9.3.3	Fungsi-fungsi Untuk List	
9.3.4	List Sebagai Parameter Fungsi	
9.4	Kegiatan Praktikum	
9.5	Latihan Mandiri	
10	Type Data Dictionary	105
10.1	Tujuan Praktikum	
10.2	Alat dan Bahan	
10.3	Materi	
10.3.1	<i>Dictionary</i> sebagai set penghitung (<i>counters</i>)	
10.3.2	<i>Dictionary</i> dan File	
10.3.3	<i>Looping</i> dan <i>Dictionary</i>	
10.3.4	<i>Advanced Text Parsing</i>	
10.4	Kegiatan Praktikum	
10.5	Latihan Mandiri	
11	Type Data Tuples	119
11.1	Tujuan Praktikum	
11.2	Alat dan Bahan	
11.3	Materi	
11.3.1	<i>Tuple Immutable</i>	
11.3.2	Membandingkan <i>Tuple</i>	
11.3.3	Penugasan <i>Tuple</i>	
11.3.4	<i>Dictionaries and Tuple</i>	
11.3.5	Multipenugasan dengan <i>dictionaries</i>	
11.3.6	Kata yang sering muncul	
11.3.7	<i>Tuple</i> sebagai kunci <i>dictionaries</i>	
11.4	Kegiatan Praktikum	
11.5	Latihan Mandiri	
12	Type Data Set	133
12.1	Tujuan Praktikum	
12.2	Alat dan Bahan	
12.3	Materi	
12.3.1	Pengenalan dan Mendefinisikan Set	
12.3.2	Pengaksesan Set	
12.3.3	Operasi-Operasi pada Set	
12.4	Kegiatan Praktikum	
12.4.1	Contoh-contoh Kasus Set	
12.5	Latihan Mandiri	

9. Tipe Data List

9.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat memahami sifat-sifat list dan dapat mendefinisikan list.
2. Dapat memahami dan melakukan operasi-operasi CRUD (Create, Read, Update dan Delete) pada list.
3. Dapat memahami dan menggunakan fungsi-fungsi Python yang berkaitan dengan list.
4. Dapat memahami dan menggunakan list sebagai parameter fungsi.

9.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

9.3 Materi

9.3.1 Sifat-sifat List

List pada Python adalah rangkaian nilai-nilai yang dapat diakses menggunakan satu nama tunggal. Apa bedanya dengan String? String adalah rangkaian dari karakter-karakter, sedangkan

list dapat berisi karakter, integer, float maupun tipe data lainnya. List juga bisa berisi list lainnya. Rangkaian nilai-nilai tersebut dituliskan di dalam [] seperti contoh berikut ini:

```
nilai_ujian = [80,75,70,90,81,84,92,71,65,80,70]
nama_pahlawan = ['Sukarno', 'Diponegoro', 'Jend. Sudirman', 'Cut Nya Dhien']
nilai_campuran = ['Javascript', 20, 34.4, True]
list_dalam_list = [23, [22, 20], 45]
```

List juga memiliki perbedaan lain dengan String, yaitu list bersifat mutable, sedangkan String bersifat immutable. Mutable artinya nilainya dapat diubah secara langsung, seperti yang ditunjukkan pada kode program berikut ini:

```
# definisikan list berisi 4 nilai
data = [10,20,30,40]
# ubah nilai index ke-0 menjadi 50
data[0] = 50
# isinya sekarang: 50, 20, 30, 40
print(data)
```

```
# definisikan sebuah string
nama = 'Yuan Lukito'
# ubah karakter index ke-0 menjadi Z
nama[0] = 'Z'
# tidak akan mencapai baris ini karena muncul error
# TypeError: 'str' object does not support item assignment
print(nama)
```

Perbedaan berikutnya, jika ada dua string yang isinya sama, keduanya menunjuk (merujuk/mengacu) pada object yang sama. Sedangkan pada list jika ada dua list dengan isi yang sama, keduanya menunjuk pada object yang berbeda. Hal tersebut dapat ditunjukkan dengan Python shell berikut ini:

```
>>> a = 'banana'
>>> b = 'banana'
>>> a is b
True

>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```


9.3.2 Mengakses dan Mengubah isi List

9.3.3 Fungsi-fungsi Untuk List

9.3.4 List Sebagai Parameter Fungsi

9.4 Kegiatan Praktikum

9.5 Latihan Mandiri

10. Tipe Data Dictionary

10.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan tentang dictionary
2. Dapat menggunakan dictionary sebagai model penghitung
3. Dapat membangun dictionary dari file
4. Dapat menggunakan perulangan dari dictionary
5. Dapat melakukan text parsing tingkat lanjut

10.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

10.3 Materi

Dictionary mempunyai kemiripan dengan *list*, tetapi lebih bersifat umum. Dalam *list*, indeks harus berupa integer sedangkan dalam dictionary indeks bisa dapat berupa apapun.

Dictionary memiliki anggota yang terdiri dari pasangan **kunci:nilai**. Kunci harus bersifat unik, tidak boleh ada dua kunci yang sama dalam satu *dictionary*.

Dictionary dapat diartikan pula sebagai pemetaan antara sekumpulan indeks (yang disebut kunci) dan sekumpulan nilai. Setiap kunci memetakan suatu nilai. Asosiasi kunci dan nilai tersebut disebut dengan pasangan nilai kunci (*key-value pair*) atau ada yang menyebutnya sebagai *item*.

Sebagai contoh, kita akan mengembangkan kamus yang memetakan dari kata-kata dalam bahasa Inggris ke bahasa Spanyol, jadi bisa dikatakan yang menjadi kunci dan nilai adalah data *string*. Asumsinya setiap kata dalam bahasa Inggris mempunyai satu arti dalam bahasa Spanyol.

Fungsi **dict** digunakan untuk membuat *dictionary* baru yang kosong. Karena *dict* merupakan *built-in function* dari python, maka penggunaanya perlu dihindari sebagai nama variabel.

```
>>> eng2sp = dict()
>>> print(eng2sp)
{}
```

Tanda kurung kurawal { } digunakan untuk merepresentasikan *dictionary* kosong. Untuk menambahkan item dalam *dictionary*, dapat menggunakan kurung kotak [].

```
>>> eng2sp['one'] = 'uno'
```

Jika kita perhatikan potongan kode diatas, dapat terlihat bahwa baris tersebut membuat item yang memetakan dari kunci 'satu' ke nilai 'uno'. Dan jika kita mencetak *dictionary* tadi, maka akan terlihat pasangan nilai kunci antara kunci dan nilai.

```
>>> print(eng2sp)
{'one': 'uno'}
```

Format output yang digunakan juga merupakan format input. Misalkan kita membuat *dictionary* baru dengan tiga item dan melakukan pencetakan hasil yang didapatkan berupa keseluruhan data dari *dictionary* tersebut.

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> print(eng2sp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

Pengurutan pasangan **kunci-nilai** tidaklah sama. Secara umum urutan item pada *dictionary* tidak dapat diprediksi. Hal ini tidaklah menjadi masalah utama karena elemen dalam *dictionary* tidak pernah diberikan **indeks dengan indeks integer**. Sebagai gantinya dapat menggunakan kunci untuk mencari nilai yang sesuai (*corresponding values*).

```
>>> print(eng2sp['two'])
'dos'
```

Kunci 'two' selalu dipasangkan dengan nilai "dos" jadi urutan pada item tidak terlalu berpengaruh. Jika menggunakan kunci yang tidak ada dalam *dictionary*, akan didapatkan pengecualian:

```
>>> print(eng2sp['four'])
KeyError: 'four'
```

Fungsi **len** pada *dictionary* digunakan untuk mengembalikan jumlah pasangan nilai kunci:

```
>>> len(eng2sp)
3
```

Operator **in** dalam *dictionary* mengembalikan nilai benar (*true*) atau salah (*false*) sesuai dengan kunci yang ada dalam *dictionary*.

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
```

Untuk mengetahui nilai yang ada pada *dictionary* dapat menggunakan method **values**. Method ini akan mengembalikan nilai sesuai dengan tipe datanya dan dikonversi kedalam *list* dan digunakan dalam operator *in*.

```
>>> vals = list(eng2sp.values())
>>> 'uno' in vals
True
```

Operator **in** menggunakan algoritma yang berbeda untuk *list* dan *dictionary*. Untuk *list* menggunakan algoritma pencarian linear. Saat *list* bertambah, waktu pencarian yang dibutuhkan menjadi lebih lama sesuai dengan panjang *list*. Dalam *dictionary*, Python menggunakan algoritma yang disebut *Hash Table* dengan properti yang luar biasa, operator **in** membutuhkan waktu yang sama untuk memprosesnya dan tidak memperdulikan banyak item yang ada didalam *dictionary*.

10.3.1 Dictionary sebagai set penghitung (*counters*)

Sebagai contoh kita diberikan sebuah *string* dan di haruskan menghitung banyaknya huruf yang muncul, beberapa cara yang bisa digunakan misalnya :

1. Membuat 26 variable untuk tiap huruf pada alfabet, kemudian memasukkanya dalam *string* untuk masing-masing karakter, menambah perhitungan yang sesuai dan menggunakan kondisional berantai.
2. Membuat *list* dengan 26 elemen, kemudian melakukan konversi setiap karakter menjadi angka (menggunakan fungsi bawaan), kemudian menggunakan angka sebagai indeks dalam *list* dan menambah perhitungan yang sesuai.
3. Membuat *dictionary* dengan karakter sebagai kunci dan perhitungan sebagai nilai yang sesuai, Karakter dapat ditambahkan item kedalam *dictionary* dan kemudian ditambahkan nilai dari item yang ada.

Dari 3 opsi diatas, kita akan melakukan perhitungan yang sama, akan tetapi dari masing-masing opsi menerapkan proses perhitungan dengan cara yang berbeda-beda.

Implementasi dilakukan dengan menggunakan perhitungan (*computation*), beberapa perhitungan biasanya lebih baik dari model perhitungan lainnya. Penggunaan komputasi model *dictionary* lebih praktis, dimana kita tidak perlu mengetahui huruf mana yang akan muncul dalam string. Kita hanya perlu memberikan ruang untuk huruf yang akan muncul. Model penerapannya sebagai berikut:

```
1 word = 'brontosaurus'
2 d = dict()
3 for c in word:
4     if c not in d:
5         d[c] = 1
6     else:
7         d[c] = d[c] + 1
8 print(d)
```

Model komputasi diatas disebut dengan *histogram*, yang mana merupakan istilah statistika dari set perhitungan (atau frekuensi).

Perulangan *for* akan melewati string. Setiap kali terjadi *looping*, jika karakter *c* tidak ada dalam *dictionary* maka akan dibuat item baru dengan kunci *c* dan nilai awal 1 (asumsinya telah muncul sekali). Jika *c* sudah ada dalam *dictionary* secara otomatis akan melakukan penambahan *d(c)*.

Output dari program diatas adalah :

```
{'b': 1, 'r': 2, 'o': 2, 'n': 1, 't': 1, 's': 2, 'a': 1, 'u': 2}
```

Histogram output menunjukkan bahwa huruf "a" dan "b" muncul sekali; "o" muncul dua kali, dan seterusnya.

Dictionary memiliki metode yang disebut *get* yang mengambil kunci dan nilai default. Jika kunci muncul di *dictionary*, akan mengembalikan nilai yang sesuai; jika tidak maka akan mengembalikan nilai default. Sebagai contoh:

```
>>> counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
>>> print(counts.get('jan', 0))
100
>>> print(counts.get('tim', 0))
0
```

Penggunaan *get* dapat dilakukan untuk menulis *loop histogram* secara lebih ringkas. Metode *get* secara otomatis menangani case dimana kunci tidak ada dalam *dictionary*. Dengan menghilangkan statement *if*, kita dapat meringkas empat baris menjadi satu baris saja.

```
1 word = 'brontosaurus'
2 d = dict()
3 for c in word:
4     d[c] = d.get(c,0) + 1
5 print(d)
```

dari program diatas adalah :

```
{'b': 1, 'r': 2, 'o': 2, 'n': 1, 't': 1, 's': 2, 'a': 1, 'u': 2}
```

Penggunaan metode *get* untuk menyederhanakan loop penghitungan ini akhirnya menjadi "idiom" yang sangat umum digunakan dalam Python. Jika dibandingkan loop menggunakan pernyataan *if* dan operator *in* dengan loop menggunakan metode *get* melakukan hal yang persis sama, tetapi yang satu lebih ringkas.

10.3.2 Dictionary dan File

Salah satu kegunaan umum *dictionary* adalah untuk menghitung kemunculan kata-kata dalam file dengan beberapa teks tertulis. Mari kita mulai dengan file kata yang sangat sederhana yang diambil dari teks Romeo dan Juliet.

Sebagai contoh pertama, akan digunakan versi teks yang disingkat dan disederhanakan tanpa tanda baca. Selanjutnya akan digunakan teks adegan dengan tanda baca yang disertakan.

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Kita akan mencoba memnulis program Python untuk membaca baris-baris file, memecah setiap baris menjadi daftar kata, dan kemudian melakukan perulangan melalui setiap kata dalam baris dan menghitung setiap kata menggunakan *dictionary*.

Asumsikan kita menggunakan dua **for** untuk perulangan. Perulangan bagian luar untuk membaca baris file dan perulangan pada dalam mekalukan iterasi melalui setiap kata pada baris tertentu. Ini adalah contoh dari pola yang disebut nested loop karena salah satu perulangan adalah perulangan bagian luar dan perulangan lainnya adalah perulangan bagian dalam.

Perulangan dalam melakukan eksekusi pada semua iterasi setiap kali perulangan bagian luar membuat iterasi tunggal. Pada bagian ini, perulangan bagian dalam iterasi "lebih cepat", sedangkan perulangan bagian luar iterasi lebih lambat.

Kombinasi dari dua perulangan bersarang memastikan bahwa ada proses perhitugnan setiap kata pada setiap baris file input.

```
1 fname = input('Enter the file name: ')
2 try:
3     fhand = open(fname)
4 except:
5     print('File cannot be opened:', fname)
6     exit()
7
8 counts = dict()
9 for line in fhand:
10     words = line.split()
11     for word in words:
12         if word not in counts:
13             counts[word] = 1
14         else:
15             counts[word] += 1
16
17 print(counts)
```

Pada statement **else** digunakan model penulisan yang lebih singkat untuk menambahkan variable. **counts[word] += 1** sama dengan **counts[word] = counts[word] + 1**. Metode lain dapat

digunakan untuk mengubah nilai suatu variabel dengan jumlah yang diinginkan, misalnya dengan menggunakan `- =`, `* =`, dan `/ =`.

Ketika kita menjalankan program, akan didapatkan data dump jumlah semua dalam urutan hash yang tidak disortir.



file romeo.txt dapat diunduh melalui moodle atau dapat menggunakan menggunakan teks Romeo Juliet pada bagian awal 10.3.2 Dictionary dan File yang kemudian disimpan dengan nama romeo.txt

```
python count1.py
Enter the file name: romeo.txt
{'and': 3, 'envious': 1, 'already': 1, 'fair': 1,
'is': 3, 'through': 1, 'pale': 1, 'yonder': 1,
'what': 1, 'sun': 2, 'Who': 1, 'But': 1, 'moon': 1,
'window': 1, 'sick': 1, 'east': 1, 'breaks': 1,
'grief': 1, 'with': 1, 'light': 1, 'It': 1, 'Arise': 1,
'kill': 1, 'the': 3, 'soft': 1, 'Juliet': 1}
```

10.3.3 Looping dan Dictionary

Dalam statement *for*, *dictionary* akan bekerja dengan cara menelusuri kunci yang ada didalamnya. *Looping* ini akan melakukan pecetakan setiap kunci sesuai dengan hubungannya.

```
1 counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
2 for key in counts:
3     print(key, counts[key])
```

Outputnya akan tampak sebagai berikut:

```
jan 100
chuck 1
annie 42
```

Output yang ditampilkan memperlihatkan bahwa kunci tidak berada dalam pola urutan tertentu. Pola ini dapat diimplementasikan dengan berbagai idiom *looping* yang telah dideskripsikan pada bagian sebelumnya. Sebagai contoh jika ingin menemukan semua entri dalam *dictionary* dengan nilai diatas 10, maka kode programnya sebagai berikut :

```
1 counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
2 for key in counts:
3     if counts[key] > 10 :
4         print(key, counts[key])
```

Pada *looping for* yang melalui kunci *dictionary*, perlu ditambahkan operator indeks untuk mengambil nilai yang sesuai untuk setiap kunci. Outputnya akan terlihat sebagai berikut :

```
jan 100
annie 42
```


Program diatas hanya akan menampilkan data nilai diatas 10.

Untuk melakukan print kunci dalam urutan secara alfabet, langkah pertama yang dilakukan adalah membuat list dari kunci pada *dictionary* dengan menggunakan method kunci yang tersedia pada object *dictionary*. Langkah selanjutnya adalah melakukan pengurutan (*sort*), *list* dan *loop* melewati *sorted list*. Langkah terakhir dengan melihat tiap kunci dan pencetak pasangan nilai *key* yang sudah diurutkan. Impelentasi dalam kode dapat dilihat dibawah ini:

```
1 counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
2 lst = list(counts.keys())
3 print(lst)
4 lst.sort()
5 for key in lst:
6     print(key, counts[key])
```

Outputnya sebagai berikut:

```
['jan', 'chuck', 'annie']
annie 42
chuck 1
jan 100
```

Pertama-tama kita melihat list dari kunci dengan tidak berurutan yang didapatkan dari method kunci. Kemudian kita melihat pasangan nilai kunci yang disusun secara berurutan menggunakan loop **for**.

10.3.4 Advenced Text Parsing

Pada bagian sebelumnya, kita menggunakan contoh file dimana kalimat pada file teersebut sudah dihilangkan tanda bacanya. Bagian ini kita akan menggukan file dengan tanda baca lengkap. Dengan menggunakan contoh yang sama yaitu romeo.txt dengan tanda baca lengkap.

```
But, soft! what light through yonder window breaks?
It is the east, and Juliet is the sun.
Arise, fair sun, and kill the envious moon,
Who is already sick and pale with grief,
```

Python sendiri mempunyai functoin **split** dimana fungsi tersebut akan mencari spasi dan mengubah kata-kata sebgau token yang dipisahkan oleh spasi. Misal pada kata "*soft!*" dan "*soft*" merupakan dua kata yang berbeda dan dalam *dictionary* adakn dibuat terpisah untuk tiap kata tersebut.

Hal tersebut juga berlaku pada penggunaan huruf kapital. Misal pada kata "*who*" dan "*Who*" dianggap sebagai kata berbeda dan dihutng secara terpisah.

Model penyelesaian lain dapat juga dengan menggunakan method *string* lain seperti **lower**, **punctuation** dan **translate**. Jika diperhatikan method string **translate** merupakan method string yang paling *stuble* (hampir tidak kentara). Berikut ini merupakan dokumentasi dari method **translate**.

```
line.translate(str.maketrans(fromstr, tostr, deletestr))
```

Ganti karakter pada *fromstr* dengan karakter pada posisi yang sama dengan *tostr* dan hapus semua karakter yang ada dalam *deletetr*. Untuk *fromstr* dan *tostr* dapat berupa string kosong dan untuk parameter pada *deletetr* dapat dihilangkan.

Kita tidak akan menentukan *tostr* tetapi kita akan menggunakan parameter *deletetr* untuk menghapus semua tanda baca. Secara otomatis Python akan memberitahu bagian mana yang dianggap sebagai "tanda baca".

```
>>> import string
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

Penggunaan dalam program sebagai berikut:

```
1 import string
2
3 fname = input('Enter the file name: ')
4 try:
5     fhand = open(fname)
6 except:
7     print('File cannot be opened:', fname)
8     exit()
9
10 counts = dict()
11 for line in fhand:
12     line = line.rstrip()
13     line = line.translate(line.maketrans('', '', string.punctuation))
14     line = line.lower()
15     words = line.split()
16     for word in words:
17         if word not in counts:
18             counts[word] = 1
19         else:
20             counts[word] += 1
21
22 print(counts)
```

Output dari program diatas adalah :

```
Enter the file name: romeo-full.txt
{'swearst': 1, 'all': 6, 'afeard': 1, 'leave': 2, 'these': 2,
'kinsmen': 2, 'what': 11, 'thinkst': 1, 'love': 24, 'cloak': 1,
a': 24, 'orchard': 2, 'light': 5, 'lovers': 2, 'romeo': 40,
'maiden': 1, 'whiteupturned': 1, 'juliet': 32, 'gentleman': 1,
'it': 22, 'leans': 1, 'canst': 1, 'having': 1, ...}
```

10.4 Kegiatan Praktikum

Kasus 10.1 Buatlah sebuah program yang dapat melakukan generate dan mentecak dictionary yang berisi angka antara 1 sampai n dalam bentuk (x,x*x)

Contoh:

Input Data = 5

Dcitionary = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

Pembahasan Kasus 1

Untuk dapat menyelesaikan kasus pertama, langkah-langkah yang menjadi penyelesaiannya adalah :

1. Membuat dan menentukan panjang dictionary nya.
2. Buat perulangan sepanjang dictionary
3. Input dictionary dengan **key = x** dan **value = x*x**.

Berikut ini contoh programnya:

```
1
2 n= int(input("Input data = "))
3 kamus = dict()
4
5 for x in range(1,n+1):
6     kamus[x]=x*x
7
8 print("Dictionary = ", kamus)
9
```

Perhatikan outputnya !!

Kasus 10.2 Buatlah program untuk mencetak semua nilai (value) unik yang ada didalam dictionary.

Contoh:

Data : [{"V": "S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"}, {"VII": "S005"}, {"V": "S009"}, {"VIII": "S007"}]

Output : Unique Values: {'S005', 'S002', 'S007', 'S001', 'S009'}

Pembahasan Kasus 2

Untuk dapat menyelesaikan kasus kedua, kita dapat menggunakan fungsi *values* pada dictionary. untuk mencari nilai unik kita ambil masing-masing nilai dari anggota dictionary, kemudiah kita kumpulkan dalam satu variabel.

Berikut ini contoh programnya:

```
1
2 data = [{"V": "S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"},
3 {"VII": "S005"}, {"V": "S009"}, {"VIII": "S007"}]
4 print("Data asli: ", data)
5 nilai_unik = set( val for dic in data for val in dic.values())
6 print("Nilai Unik: ", nilai_unik)
7
```

Perhatikan outputnya !!

Kasus 10.3 Dengan menggunakan file **words.txt**, buatlah program untuk menyimpan kunci (keys) pada dictionary. Tambahkan pengecekan apakah suatu kata yang diinputkan ada didalam daftar tersebut. Jika ada silakan cetak kata ditemukan, jika tidak ada silakan cetak kata tidak ditemukan.

Silakan cek bagian dibawah ini untuk contoh output dari programnya.

```
Masukkan nama file : words.txt
Kata yang dicari : programming
```

Daftar Kamus

```
{'Writing': 1, 'programs': 2, 'or': 3, 'programming':
4, 'is': 5, 'a': 6, 'very': 7, 'creative': 8, 'and':
9, 'rewarding': 10, 'activity': 11, 'You': 12,
'can': 13, 'write': 14, 'for': 16, 'many': 17,
'reasons': 18, 'ranging': 19, 'from': 20, 'making':
21, 'your': 22, 'living': 23, 'to': 24, 'solving':
25, 'difficult': 27, 'data': 28, 'analysis': 29, }
```

Kata programming ditemukan dalam kamus

Pembahasan Kasus 3

Untuk dapat membuat program diatas, langkah-langkah yang harus dilakukan adalah :

1. Buat program untuk membaca file txt.
2. Input kata yang dicari.
3. Buat dictionary baru untuk menyimpan file yang sudah dibaca.
4. simpan file dalam dictionary, jangan lupa cek duplikasinya.
5. Cetak dictionary nya.
6. Lakukan pengecekan terhadap kata yang diinputkan.
7. Cetak hasilnya.

Berikut adalah kode programnya.

```
1
2 count = 0
3 dictionary_words = dict()
4 fname = input('Masukkan nama file : ')
5 fword = input('Kata yang dicari : ')
6 try:
7     fhand = open(fname)
8 except FileNotFoundError:
9     print('File tidak bisa dibuka !!', fname)
10    exit()
11
12 for line in fhand:
13     words = line.split()
14     for word in words:
```

```

15         count += 1
16         if word in dictionary_words:
17             continue # cek duplikasi
18         dictionary_words[word] = count # kata yg pertama muncul
19
20 print('\nDaftar Kamus : \n')
21 print(dictionary_words)
22
23 if fword in dictionary_words:
24     print('\nKata %s ditemukan dalam kamus' % fword)
25 else:
26     print('\nKata %s tidak ditemukan dalam kamus' % fword)
27
28

```

Silahkan lihat hasil programnya!

Kasus 10.4 Buat program yang dapat membagi kategori dari setiap email yang masuk dimana pada hari itu perintah **commit** dilakukan. Kemudian hitunglah berapa banyak hari dimana perintah **commit** itu sering dilakukan. Gunakan file mbox-short.txt untuk mendapatkan kategori dari setiap email yang masuk tersebut.

Pembahasan Kasus 4

Untuk dapat menyelesaikan kasus, langkah-langkah yang harus dilakukan adalah :

1. Cek pola yang ada
2. Buat program untuk membaca file txt.
3. Buat dictionary
4. Cari baris yang dimulai dengan **"From"**
5. Cari kata ketiga, lakukan perhitungan berjalan untuk menghitung setiap hari dalam 1 minggu.
6. Simpan dalam dictionary.
7. Cetak dictionary nya.

Untuk melakukan pengecekan pola, silakan buka file mbox-short.txt. File tersebut merupakan standart format yang berisi *multiple mail message*. Baris pertama dimulai dengan **"From"** untuk memisahkan antar *message*. Informasi mengenai format mbox silakan cek <https://en.wikipedia.org/wiki/Mbox>.

Pengecekan pola

Baris contoh :

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

Program:

python kasus4.py

Enter a file name: mbox-short.txt

{'Fri': 20, 'Thu': 6, 'Sat': 1}

Berdasarkan pengecekan yang telah kita lakukan diatas, dapat dibuat kode programmnya sebagai berikut:

```

1
2 dictionary_days = dict() # dictionary baru
3 fname = input('Enter a file name: ')
4 try:

```

```

5     fhand = open(fname)
6 except FileNotFoundError:
7     print('File cannot be opened:', fname)
8     exit()
9
10    for line in fhand:
11        words = line.split()
12        if len(words) < 3 or words[0] != 'From':
13            continue
14        else:
15            if words[2] not in dictionary_days:
16                dictionary_days[words[2]] = 1           # pertama
17            else:
18                dictionary_days[words[2]] += 1          # +1
19
20    print(dictionary_days)
21

```

Silahkan lihat hasil programnya!

10.5 Latihan Mandiri

Latihan 10.1 Bualah sebuah program untuk mendapatkan nilai key, value, dan item dari sebuah dictionary.

Contoh:

Dictionary : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

Output:

key	value	item
1	10	1
2	20	2
3	30	3
4	40	4
5	50	5
6	60	6

Latihan 10.2 Bualah sebuah program untuk memetakan dua list menjadi satu dictionary.

Contoh:

Data List

```
Lista = ['red', 'green', 'blue']
```

```
Listb = ['#FF0000', '#008000', '#0000FF']
```

Output

```
{'green': '#008000', 'blue': '#0000FF', 'red': '#FF0000'}
```

Latihan 10.3 Dengan menggunakan file **mbox-short.txt**, buatlah program yang dapat membaca log email dan sajikan dalam histogram menggunakan dictionary. Kemudian hitung berapa banyak pesan yang masuk dari email dan sajikan dalam bentuk dictionary.

Silakan cek bagian dibawah ini untuk contoh output dari programnya.

Masukkan nama file : mbox-short.txt

```
{'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3,
'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1,
'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3,
'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1,
'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2,
'ray@media.berkeley.edu': 1}
```

Latihan 10.4 Dengan menggunakan file **mbox-short.txt**, buat program untuk mencatat data nama domain pengirim pesan. Hitunglah jumlah pesan yang dikirim masing-masing domain. sajikan dalam bentuk dictionary.

Silakan cek bagian dibawah ini untuk contoh output dari programnya.

Masukkan nama file: mbox-short.txt

```
{'media.berkeley.edu': 4, 'uct.ac.za': 6, 'umich.edu': 7,
'gmail.com': 1, 'caret.cam.ac.uk': 1, 'iupui.edu': 8}
```


11. Tipe Data Tuples

11.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan tentang tuple.
2. Dapat membandingkan tuple.
3. Dapat memberikan penugasan pada tuple.
4. Dapat menjelaskan hubungan antara tuple dan dictionary.
5. Dapat memberikan multi penugasan dengan dictionary.
6. Dapat menggunakan tuple sebagai kunci dalam dictionary.

11.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

11.3 Materi

11.3.1 Tuple Immutable

Tuple bisa dikatakan menyerupai *list*. Nilai yang disimpan dalam *tuple* dapat berupa apapun dan diberikan indeks bilangan bulat (*integer*). Perbedaan utama adalah sifat *tuple* yang *immutable*, yaitu dimana anggota dalam *tuple* tidak bisa dirubah. *Tuple* sendiri dapat dibandingkan (*compare*) dan bersifat *hashable* sehingga dapat dimasukkan dalam *list* sebagai *key* pada *dictionary* python.

Sebuah objek disebut *hashable* jika memiliki nilai *hash* yang tidak pernah berubah selama hidupnya (menggunakan method `__hash__()`), dan dapat dibandingkan dengan benda-benda lain (menggunakan method `__eq__()` atau `__cm__()`). Obyek *Hashable* membandingkan yang sama dengan memiliki nilai *hash* yang harus sama.

Hashability membuat sebuah objek yang dapat digunakan sebagai kamus kunci dan mengatur anggota, karena struktur data ini menggunakan nilai *hash* secara internal.

Objek pada python *built-in* biasanya *hashable*, sementara tidak bisa berubah wadah (seperti daftar atau kamus) adalah. Objek yang merupakan *instance* dari kelas yang didefinisikan pengguna yang *hashable* secara default; mereka semua membandingkan yang tidak sama, dan mereka nilai *hash* adalah `id()`.

selengkapnya : <https://docs.python.org/3/glossary.html>

Sintaks penulisan *tuple* :

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Sintaks lain penulisan *tuple* dapat menggunakan tanda kurung:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Tuple dengan satu element, ditambahkan koma (,) dibelakang penulisan *tuple*.

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

Jika tidak menambahkan tanda koma, akan dianggap sebagai *string*.

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
```

Model sintaks lain dengan menggunakan fungsi ***built-in tuple***. Ketika tidak diisi dengan argument apapun, secara langsung akan membentuk *tuple* kosong.

```
>>> t = tuple()
>>> print(t)
()
```

Jika argumennya berupa urutan (*string*, *list*, atau *tuple*), akan mengembalikan nilai *tuple* dengan elemen-elemen yang berurutan.

```
>>> t = tuple('duta wacana')
>>> print(t)
('d', 'u', 't', 'a', 'w', 'a', 'c', 'a', 'n', 'a')
```

Tuple merupakan nama dari **constructor**, kita tidak bisa menggunakannya sebagai nama variabel. Sebagian besar operator yang bekerja pada *list* bekerja juga pada *tuple*. Tanda kurung kotak [] menandakan indeks element dari *tuple*.

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print(t[0])
'a'
```

Untuk menampilkan rentang nilai dari element *tuple* dapat menggunakan:

```
>>> print(t[1:3])
('b', 'c')
```

Tuple bersifat *immutable* artinya element pada *tuple* tidak dapat berubah. Jika anda berusaha mengubah *tuple*, maka akan didapatkan error.

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

Element pada *tuple* tidak dapat dirubah tetapi dapat diganti dengan elemen lain.

```
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```

11.3.2 Membandingkan *Tuple*

Operator perbandingan (*compare*) dapat bekerja pada *tuple* dan model sekuensial lainnya (*list*, *dictionary*, *set*). Cara kerjanya dengan membandingkan elemen pertama dari setiap sekuensial yang ada. Jika ditemukan adanya kesamaan, akan berlanjut ke elemen berikutnya. Proses ini berlangsung terus menerus hingga ditemukan adanya perbedaan.

```
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
```

Fungsi (*sort*) pada python bekerja dengan cara yang sama. Tahap pertama akan melakukan pengurutan berdasarkan elemen pertama. Pada kasus tertentu akan mengurutkan berdasarkan elemen kedua dan sebagainya. Fitur ini disebut dengan DSU – (*Decorate, Sort, Undercorate*)

1. **Decorate** – urutan (sekuensial) membangun daftar *tuple* dengan satu atau lebih *key* pengurutan sebelum elemen dari urutan (sekuensial).
2. **Sort** – *list tuple* menggunakan *sort* (fungsi bawaan di python).
3. **Undercorate** – melakukan ekstraksi pada elemen yang telah diurutkan pada satu sekuensial.

Untuk memahaminya, silakan perhatikan contoh dibawah ini. Kita mempunyai sebuah kalimat dan akan melakukan pengurutan kata dalam kalimat tersebut dari yang paling panjang ke yang paling pendek.

```
1
2 kalimat = 'but soft what light in yonder window breaks'
3 dafkata = kalimat.split()
4 t = list()
5 for kata in dafkata:
6     t.append((len(kata), kata))
7
8 t.sort(reverse=True)
9
10 urutan = list()
11 for length, kata in t:
12     urutan.append(kata)
13
14 print(urutan)
15
16
```

Looping yang pertama akan membuat daftar *tuple*, yang berisi daftar kata sesuai dengan panjangnya. Fungsi *sort* akan membandingkan elemen pertama dari list dari panjang kata yang ada dan kemudian akan menuju ke elemen kedua jika kondisi sesuai. *Keyword reverse=True* digunakan untuk melakukan urutan secara terbalik (*descending*).

Looping kedua akan membangun daftar *tuple* dalam sesuai urutan alfabet diurutkan berdasarkan panjangnya. Pada contoh diatas Empat kata dalam kalimat akan diurutkan secara alfabet terbalik. Kata "what" akan muncul sebelum "soft" didalam *list*.

Output program diatas dapat dilihat dibawah ini:

```
['yonder', 'window', 'breaks', 'light', 'what',
'soft', 'but', 'in']
```

Kata-kata yang muncul diurutkan sebagai *list* dengan urutan panjang kata dari yang paling panjang ke kata yang paling pendek.

11.3.3 Penugasan *Tuple*

Salah satu fitur unik dari Python adalah kemampuannya untuk memiliki *tuple* disisi kiri dari statement penugasan. Hal ini mengijinkan untuk menetapkan lebih dari satu variabel pada sisi sebelah kiri secara berurutan.

Contohnya ketika ada dua daftar elemen yang merupakan urutan. Kita akan mencoba menetapkan elemen pertama dan kedua dari urutan tersebut kedalam variabel *x* dan *y* dalam satu *statement*.

```
>>> m = [ 'have', 'fun' ]
>>> x, y = m
>>> x
'have'
>>> y
'fun'
>>>
```

Python akan menterjemahkan sintaks *tuple* dalam langkah-langkah sebagai berikut:

```
>>> m = [ 'have', 'fun' ]
>>> x = m[0]
>>> y = m[1]
>>> x
'have'
>>> y
'fun'
>>>
```

Tuple yang digunakan pada contoh diatas menggunakan statement penugasan disebelah kiri dan tidak menggunakan tanda kurung. Berikut ini merupakan contoh yang sama dengan menggunakan tanda kurung (*parentheses*).

```
>>> m = [ 'have', 'fun' ]
>>> (x, y) = m
>>> x
'have'
>>> y
'fun'
>>>
```

Tuple memungkinkan pula untuk menukar nilai variabel dalam satu statement.

```
>>>> a, b = b, a
```

Dari contoh diatas, dapat dilihat bahwa kedua statemnt merupakan *tuple*. Bagian kiri merupakan *tuple* dari variable dan bagian kanan merupakan tuple dari *expresions*. Tiap nilai pada bagian kanan diberikan/ditugaskan ke masing-masing variable di sebelah kiri. Semua *expresions* pada bagian kiri dievaluasi sebelum diberikan penugasan.

Yang perlu diperhatikan adalah jumlah variabel antara sisi kiri dan sisi kanan harus sama.

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

Pada sisi sebelah kanan biasanya terdapat data sekuensial *string*, *list* atau *tuple*. Sebagai contoh, kita akan membagi alamat email menjadi *user name* dan *domain* seperti berikut ini.

```
>>> email = 'didanendya@ti.ukdw.ac.id'
>>> username, domain = email.split('@')
```

Nilai yang dikembalikan dari **split** terdiri dari dua elemen yang dipisahkan tanda "@", elemen pertama berisi **username** dan elemen selanjutnya berisi **domain**.

```
>>> print( usernam)
didanendya
>>> print( domain)
ti.ukdw.ac.id
```

11.3.4 Dictionaries and Tuple

Dictionaries mempunyai metode yang disebut **items** untuk mengembalikan nilai *list* dari *tuple* dimana tiap *tuple*-nya merupakan *key-value pair* (pasangan kunci dan nilai).

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = list(d.items())
>>> print(t)
[('b', 1), ('a', 10), ('c', 22)]
```

Seperti pada *dictionary* umumnya, *item* yang dikembalikan nilainya tidak berurutan. Bagaimana pun juga, *list* dari *tuple* adalah **list**, dan *tuple* membandingkannya sehingga kita dapat melakukan pengurutan (*sort*) pada *tuple*. Melakukan konversi *dictionary* dari *list tuple* merupakan cara untuk menampilkan isi *dictionary* yang diurutkan berdasarkan kuncinya.

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = list(d.items())
>>> t
[('b', 1), ('a', 10), ('c', 22)]
>>> t.sort()
>>> t
[('a', 10), ('b', 1), ('c', 22)]
```

List yang muncul diurutkan secara *ascending* berdasarkan alfabet dan *key value*.

11.3.5 Multipenugasan dengan dictionaries

Kombinasi antara **items**, **tuple assignment** dan **for** akan menghasilkan kode tertentu pada *keys* dan *values* dari *dictionary* dalam satu loop.

```
for key, val in list(d.items()):
    print(val, key)
```

Pada bagian looping ini, terdapat 2 iterasi variabel karena *item* mengembalikan *list* dari *tuple* dan **key**. **val** merupakan penugasan *tuple* yang melakukan iterasi berulang melalui pasangan *key-value* pada *dictionary*.

Pada setiap iterasi yang melalui loop, baik **key** dan **value** akan bergerak maju menuju pasangan *key-value* berikutnya pada *dictionary* (masih dalam urutan *hash*).

Output dari looping diatas

```
10 a
22 c
1 b
```

Jika dilakukan penggabungan dari kedua teknik diatas, kita dapat melakukan pencetakan isi *dictionary* yang diurutkan berdasarkan nilai yang disimpan di setiap pasangan *key-value*.

Untuk melakukannya langkah pertama dengan membuat *list tuple* di mana masing-masing *tuple* beranggotakan (*value, key*). Method *item* akan memberikan list *tupel (value, key)* dan akan melakukan pengurutan berdasarkan *value*, bukan *key*. Setelah daftar *list* dengan *value-key* tuple terbentuk, langkah selanjutnya mengurutkan *list* tersebut dengan urutan terbalik dan melakukan pencetakan *list* baru yang disortir.

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> l = list()
>>> for key, val in d.items() :
...     l.append( (val, key) ) ...
>>> l
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> l.sort(reverse=True)
>>> l
[(22, 'c'), (10, 'a'), (1, 'b')]
>>>
```

Dengan melakukan penyusunan *list* dari *tuple* yang memiliki *value* sebagai elemen pertama, akan mudah untuk mengurutkan *list* dari *tuple* tersebut dan mendapatkan isi *dictionary* yang diurutkan berdasarkan nilainya.

11.3.6 Kata yang sering muncul

Pada bagian ini kita akan mencoba menampilkan kata yang sering muncul pada text dari *Romeo and Juliet Act 2, Scene 2*. Silakan download *romeo-full.txt* pada bagian materi. Kita akan coba membuat program menggunakan *list* dari *tuple* untuk menampilkan 10 kata yang paling sering muncul.

```
1
2 import string
3 fhand = open('romeo-full.txt')
4 counts = dict()
5 for line in fhand:
6     line = line.translate(str.maketrans('', '', string.punctuation))
7     line = line.lower()
8     words = line.split()
9     for word in words:
10         if word not in counts:
11             counts[word] = 1
12         else:
13             counts[word] += 1
14
15 # urutkan dictionary by value
16 lst = list()
17 for key, val in list(counts.items()):
18     lst.append((val, key))
19
```

```

20 lst.sort(reverse=True)
21
22 for key, val in lst[:10]:
23     print(key, val)
24

```

Bagian pertama dari program digunakan untuk membaca file dan melakukan komputasi terhadap *dictionary* yang memetakan tiap kata ke sejumlah kata dalam dokumen yang tidak berubah. Dengan menambahkan *print out counts*, list tuple (*val, key*) dibuat dan diurutkan dalam list dengan urutan terbalik.

Nilai pertama akan digunakan dalam perbandingan. Jika ada lebih dari satu *tuple* dengan nilai yang sama, kemudian akan melihat ke elemen kedua (*key*), sehingga *tuple* di mana nilainya yang sama akan diurutkan berdasarkan urutan abjad sesuai *key*.

Pada bagian akhir, ada looping yang melakukan iterasi penugasan ganda dan mencetak 10 kata yang paling sering muncul dengan melakukan perulangan pada list **lst[:10]**:

Secara lengkap output dari program diatas akan menampilkan kata yang sesuai untuk frekuensi analisis.

```

61 i
42 and
40 romeo
34 to
34 the
32 thou
32 juliet
30 that
29 my
24 thee

```

Penguraian dan analisis data yang kompleks ini dapat dilakukan dengan 19 baris program python dipahami. Hal tersebut merupakan salah satu alasan mengapa python adalah pilihan yang baik sebagai bahasa untuk mengeksplorasi informasi.

11.3.7 Tuple sebagai kunci *dictionaries*

Tuple merupakan *hashable* dan *list* tidak. Ketika kita ingin membuat *composite key* yang digunakan dalam *dictionary*, kita dapat menggunakan tuple sebagai *key*.

Misalnya menggunakan *composite key* jika ingin membuat direktori telepon yang memetakan dari pasangan *last-name, first-name* ke nomor telepon Dengan asumsi bahwa kita telah mendefinisikan variabel *last*, *first*, dan nomor. Penulisan pernyataan penugasan dalam *dictionary* sebagai berikut:

```
directory[last,first] = number
```

Expression yang ada didalam kurung kotak adalah *tuple*. Langkah selanjutnya dengan memberikan penugasan *tuple* pada looping **for** yang berhubungan dengan *dictionary*.


```
for last, first in directory:
    print(first, last, directory[last,first])
```

Looping yang berhubungan dengan *keys* pada direktori diatas merupakan *tuple*. Elemen dari masing-masing *tuple* ditetapkan pertama kali, kemudian dilakukan pencetakan nama dan nomor telepon yang sesuai.

```
>>> last = 'nendya'
>>> first = 'dida'
>>> number = '088112266'
>>> directory = dict()
>>> directory[last, first] = number
>>> for last, first in directory:
...     print(first, last, directory[last,first])
...
dida nendya 088112266
>>>
```

11.4 Kegiatan Praktikum

Kasus 11.1 Buatlah program yang dapat melakukan proses berikut:

1. Membuat dan mencetak tuple.
2. Membagi tuple kedalam string.
3. Membuat tuple yang berisi string dari sebuah kata.
4. Membuat tuple yang berisi semua, kecuali huruf pertama dari sebuah string
5. Mencetak tuple secara terbalik

Contoh:

```
kota: ('Jakarta', 'Jogja', 'Surabaya')
kota[0]: Jakarta
kota[1]: Jogja
kota[2]: Surabaya

str1: Jakarta
str2: Jogja
str3: Surabaya

tpl: ('B', 'e', 'l', 'a', 'j', 'a', 'r', ' ', 'P', 'y', 't', 'h', 'o', 'n')

tpl1: ('e', 'l', 'a', 'j', 'a', 'r', ' ', 'P', 'y', 't', 'h', 'o', 'n')

urutan: ('Jaka', 'Joko', 'Jono')
urutan_terbalik: ('Jono', 'Joko', 'Jaka')
```

Pembahasan Kasus 11.1

Program diatas merupakan fungsi dasar tuple pada pemrograman python. Untuk lebih jelasnya silakan perhatikan kode program berikut ini.

```

1
2 '''
3 1) Membuat dan mencetak tuple.
4 '''
5 kota = ("Jakarta", "Jogja", "Surabaya")
6 print("kota: ", kota)
7 print("kota[0]: ", kota[0])
8 print("kota[1]: ", kota[1])
9 print("kota[2]: ", kota[2])
10 print() # prints baris
11
12 '''
13 2) Membagi tuple kedalam string.
14 '''
15 str1, str2, str3 = kota
16 print("str1: ", str1)
17 print("str2: ", str2)
18 print("str3: ", str3)
19 print()
20
21 '''
22 3) Membuat tuple yang berisi string dari sebuah kata.
23 '''
24 tpl = tuple("Belajar Python")
25 print("tpl: ", tpl)
26 print()
27
28 '''
29 4) Membuat tuple yang berisi semua, kecuali huruf pertama dari sebuah string
30 '''
31 # direct string
32 tpl1 = tuple("Belajar Python"[1:])
33 print("tpl1: ", tpl1)
34
35
36 '''
37 5) Mencetak tuple secara terbalik
38 '''
39 name = ("Jaka", "Joko", "Jono")
40 # using slicing technique
41 rev_name = name[::-1]
42 rev2= name[::-2]
43 print("urutan: ", name)
44 print("urutan_terbalik: ", rev_name)

```

Kasus 11.2 Buatlah program yang dapat menghitung jumlah commit yang dilakukan oleh akun email tertentu. Gunakan file mbox-short.txt atau mbox.txt

Enter a file name: mbox-short.txt

```
cwen@iupui.edu 5
```

```
Enter a file name: mbox.txt
```

```
zqian@umich.edu 195
```

Pembahasan Kasus 11.2

Untuk mengerjakan kasus 11.2 beberapa langkah yang harus dilakukan adalah :

1. Membaca file text
2. Parsing **From** dan line dan hitung pesan yang ada dalam dictionary.
3. Buat list untuk mencetak tuple (email, jumlah).
4. Gunakan urutan terbaik untuk mencetak email yang paling banyak melakukan commit.

Gunakan pola : From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
untuk parsingnya.

Berdasarkan pengecekan yang telah kita lakukan diatas, dapat dibuat kode programnya sebagai berikut:

```
1
2 dic_email = dict()           # variables
3 lst = list()
4
5 fname = input('Nama File: ')
6 try:
7     fhand = open(fname)
8 except FileNotFoundError:
9     print('File tidak bisa dibuka:', fname)
10    quit()
11
12 for baris in fhand:
13     kata = baris.split()
14     if len(kata) < 2 or kata[0] != 'From':
15         continue
16     else:
17         if kata[1] not in dic_email:
18             dic_email[kata[1]] = 1           # entri 1
19         else:
20             dic_email[kata[1]] += 1         # +1
21
22 for key, val in list(dic_email.items()):
23     lst.append((val, key))                  # isi daftar dengan nilai kunci dict
24
25 lst.sort(reverse=True)                    # sort nilai tertinggi
26
27 for key, val in lst[:1]:                  # tampilkan nilai pertama (terbesar)
28     print(val, key)
29
30
```

Silahkan lihat hasil programnya!

11.5 Latihan Mandiri

Latihan 11.1 Buatlah program untuk melakukan pengecekan apakah semua anggota yang ada didalam tuple sama.

Contoh:

```
tA= (90, 90, 90, 90)
```

Output

```
True
```

Latihan 11.2 Buatlah program dengan menggunakan tuple yang dapat melakukan proses seperti pada kasus 11.1, Gunakan data diri anda masing-masing dan lakukan perubahan supaya didapatkan output seperti contoh berikut ini :

Contoh:

```
Data: ('Matahari Bhakti Nendya', '22064091', 'Bantul, DI Yogyakarta')
```

```
NIM      : 22064091
```

```
NAMA     : Matahari Bhakti Nendya
```

```
ALAMAT   : Bantul, DI Yogyakarta
```

```
NIM: ('2', '2', '0', '6', '4', '0', '9', '1')
```

```
NAMA DEPAN: ('a', 't', 'a', 'h', 'a', 'r', 'i')
```

```
NAMA TERBALIK: ('Nendya', 'Bhakti', 'Matahari')
```

Latihan 11.3 Buatlah program untuk menghitung distribusi jam dalam satu hari dimana ada pesan yang diterima dari setiap email yang masuk. Gunakan file mbox-short.txt untuk sebagai datanya. Berikut ini adalah contoh output dari programnya.

Contoh:

Enter a file name: mbox-short.txt

04 3

06 1

07 1

09 2

10 3

11 6

14 1

15 2

16 4

17 2

18 1

19 1

12. Tipe Data Set

12.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat melakukan operasi-operasi dasar pada tipe data Set, seperti mendefinisikan, mengakses, mengubah, menghapus dan melakukan pencarian data pada Set
2. Dapat melakukan operasi-operasi yang melibatkan dua set atau lebih, seperti union, difference, intersection dan symmetric difference

12.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

12.3 Materi

12.3.1 Pengenalan dan Mendefinisikan Set

Set adalah salah satu tipe data pada Python yang dapat digunakan untuk menyimpan sekumpulan data yang semuanya unik. Biasa juga dikenal dengan istilah himpunan. Beberapa sifat Set pada Python adalah:

- Isi dari Set disebut sebagai anggota (member).

- Anggota dari Set harus bersifat immutable. Beberapa tipe data immutable pada Python: integer, float, string, tuple dan lain-lain. Dengan demikian list dan dictionary (mutable) tidak dapat dimasukkan ke dalam Set.
- Set sendiri bersifat mutable, artinya anda dapat menambah atau mengurangi isi dari sebuah Set. Karena itu Set tidak dapat dimasukkan ke dalam Set.

Untuk mendefinisikan Set, ada beberapa cara yang dapat dilakukan yaitu dengan menggunakan notasi {} dan fungsi set() seperti contoh berikut ini:

```
# dengan menggunakan {}
bilangan_genap = {2, 4, 6, 8, 10, 12}
bilangan_ganjil = {1, 3, 5, 7, 9, 11}

# dengan menggunakan fungsi set()
pernah_ke_bulan = set('Neil Armstrong', 'Buzz Aldrin')
```

Untuk mendefinisikan Set kosong anda tidak dapat menggunakan notasi {}, tetapi harus menggunakan fungsi set() seperti contoh berikut:

```
# dengan fungsi set()
pernah_ke_mars = set()    # menghasilkan set kosong

data = {}                # ini akan menghasilkan dictionary kosong
```

12.3.2 Pengaksesan Set

Set tidak memiliki indeks, karena itu anda tidak dapat mengakses anggota-anggota dari sebuah Set secara langsung menggunakan indeks. Perhatikan contoh program berikut ini:

```
nim = {'71200120', '71200195', '71200214'}
jumlah_nim = len(nim)
print(jumlah_nim)          # akan menghasilkan output 3

# tampilkan isi set satu-persatu
for n in nim:
    print(n)
```

Output yang dihasilkan dari program tersebut adalah sebagai berikut:

```
3
71200214
71200195
71200120
```


Jika dicermati, output yang dihasilkan urutannya berbeda dengan deklarasi Set sebelumnya. Hal ini disebabkan karena pada Set tidak ada indeks, sehingga tidak ada urutan posisi anggota. Pada tipe data Set, posisi anggota tidaklah penting.

Set adalah tipe data mutable, artinya isinya bisa bertambah atau berkurang. Program berikut ini menunjukkan bagaimana cara menambah anggota sebuah Set dengan fungsi add():

```
# definisikan sebuah set kosong
plat_nomor = set()

# tambahkan plat nomor 'AB 1890 XA'
plat_nomor.add('AB 1890 XA')

# tambahkan plat nomor 'AD 6810 MT'
plat_nomor.add('AD 6810 MT')

# jumlah anggota di dalam Set
print(len(plat_nomor))

# tambahkan plat yang sama sekali lagi
plat_nomor.add('AB 1890 XA')

# tampilkan semua plat nomor
for plat in plat_nomor:
    print(plat)
```

Output yang dihasilkan adalah sebagai berikut:

```
2
AD 6810 MT
AB 1890 XA
```

Set memiliki mekanisme untuk melakukan pengecekan apakah anggota baru yang akan dimasukkan sudah ada di dalam Set (cek duplikasi). Jika belum ada, maka anggota tersebut bisa masuk ke dalam Set. Tetapi jika ternyata di dalam Set sudah ada anggota dengan nilai yang sama, maka pemanggilan fungsi add() tidak akan menambah anggota dari Set. Pengecekan duplikasi ini sudah ada di dalam fungsi add(), sehingga anda tidak perlu melakukannya sendiri.

Untuk menghapus anggota dari sebuah Set, ada beberapa cara yaitu dengan fungsi discard(), remove(), pop() dan clear(). Perbedaan dari empat fungsi tersebut dapat dilihat pada Gambar 12.1.

Berikut ini adalah contoh program yang mendemonstrasikan penghapusan anggota dari sebuah Set:

```
bilangan_prima = {13, 23, 7, 29, 11, 5}

# hapus 5 dari set tersebut
bilangan_prima.remove(5)
```

discard()	remove()	pop()	clear()
Menghapus satu elemen yang disebutkan	Menghapus satu elemen yang disebutkan	Mengambil salah satu dan menghapusnya dari set (tidak tentu)	Menghapus seluruh elemen di dalam set
Tidak ada error	Muncul error jika elemen yang dihapus tidak ada	Error jika set kosong	Tidak ada error

Gambar 12.1: Fungsi-fungsi untuk menghapus anggota dari Set.

```
print(bilangan_prima)

# hapus 97 (tidak ada)
bilangan_prima.discard(97)
print(bilangan_prima)

# ambil dan hapus salah satu
bilangan = bilangan_prima.pop()
print(bilangan)
print(bilangan_prima)

# kosongkan set
bilangan_prima.clear()
print(bilangan_prima)
```

Output yang dihasilkan dari program tersebut adalah:

```
{5, 7, 11, 13, 23, 29}    # awal
{7, 11, 13, 23, 29}      # setelah 5 dihapus. Perhatikan urutan berubah
{7, 11, 13, 23, 29}      # 97 tidak ada, sehingga Set tidak berubah
7                          # fungsi pop() mengeluarkan 7
{11, 13, 23, 29}         # setelah 7 keluar dari Set
set()                    # setelah isi dari Set dihapus semua
```

Fungsi `discard()` tidak akan menghasilkan error jika anggota yang ingin dihapus tidak ada di dalam Set. Sedangkan fungsi `pop()` akan mengambil salah satu anggota (secara acak) dan mengeluarkannya dari Set. Fungsi `pop()` akan berguna jika kita ingin memproses isi dari Set satu-persatu tanpa memperdulikan urutan/posisi dari setiap anggota di dalam Set.

Bagaimana jika anda ingin mengubah nilai salah satu anggota di dalam Set? Pada Set anda tidak bisa mengubah langsung nilai dari salah satu anggota di dalam Set. Yang dapat dilakukan adalah melakukan operasi penggantian (`replace`) dengan cara menghapus anggota yang mau diubah,

kemudian memasukkan anggota baru yang nilainya sesuai dengan yang diinginkan. Contohnya dapat dilihat pada program berikut ini:

```
# Buat Set dari List
ikan = set(['koi', 'koki', 'kembung', 'salmon'])
print(ikan)

# ganti koi menjadi teri
ikan.remove('koi')
ikan.add('teri')
print(ikan)
```

Output yang dihasilkan dari program tersebut adalah sebagai berikut:

```
{'koi', 'koki', 'salmon', 'kembung'}
{'teri', 'salmon', 'kembung', 'koki'}
```

Untuk mengubah nilai 'koi' menjadi 'teri', yang perlu dilakukan adalah hapus dulu anggota 'koi', kemudian masukkan anggota baru dengan nilai 'teri'. Sekali lagi, yang perlu ditekankan di sini adalah pada Set tidak mengenal adanya posisi/urutan data. Sehingga output dari program tersebut nilai 'koi' sudah tidak ada, digantikan oleh nilai 'teri'. Kemudian urutan isi Set ikan saat ditampilkan tidaklah sama (ada perubahan). Setiap kali ada operasi penambahan dan penghapusan maka urutan anggota di dalam Set biasanya akan berubah.

12.3.3 Operasi-Operasi pada Set

Operasi-operasi pada Set adalah operasi-operasi pada himpunan. Berikut ini adalah daftar operasi-operasi Set pada Python:

- **Operator Union.** Menggabungkan dua Set menjadi satu. Dapat menggunakan operator `|` maupun fungsi `union()`.
- **Operator Intersection.** Menghasilkan irisan dari dua Set. Dapat menggunakan operator `&` maupun fungsi `intersection()`.
- **Operator Difference.** Menghasilkan Set baru yang merupakan selisih dari dua Set yang dibandingkan. Dapat menggunakan operator `-` maupun fungsi `difference()`.
- **Operator Symmetric Difference.** Menghasilkan Set baru yang merupakan jumlah dari dua Set kecuali irisannya. Dapat menggunakan operator `^` maupun fungsi `symmetric_difference()`.

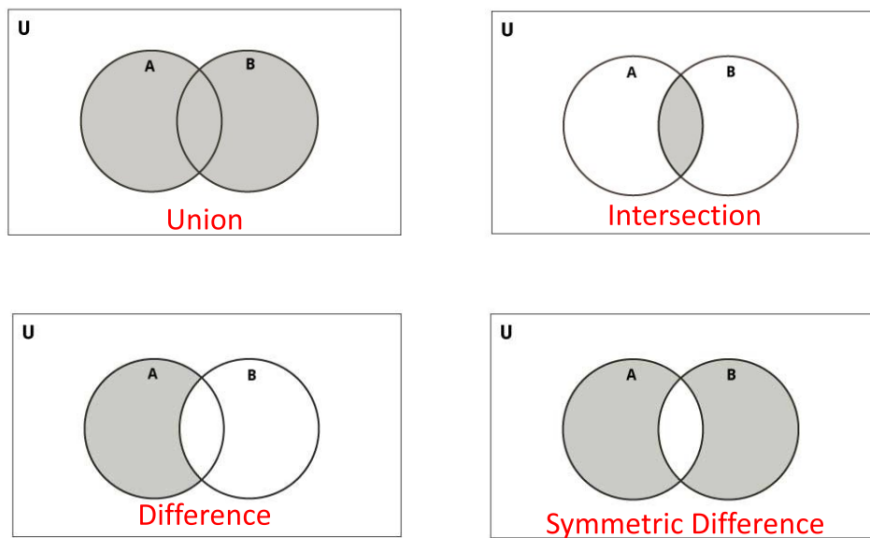
Ilustrasi dari operasi-operasi tersebut dapat dilihat pada Gambar 12.2.

Operator Union

Contoh penggunaan operator union dapat dilihat pada contoh program berikut ini:

```
merek_hp = {'Samsung', 'Apple', 'Xiaomi', 'Sony'}
merek_ac = {'LG', 'Samsung', 'Panasonic', 'Daikin', 'Sony'}

# union dari merek_hp dan merek_ac
gabungan = merek_hp | merek_ac
```



Gambar 12.2: Operasi-operasi pada Set.

```
# bisa juga menggunakan gabungan = merek_hp.union(merek_ac)
print(gabungan)
```

Output yang dihasilkan dari program tersebut adalah anggota dari Set merek_hp digabungkan dengan anggota dari Set merek_ac, menghasilkan Set baru bernama gabungan. Ada duplikasi (anggota yang sama) yaitu 'Samsung' dan 'Sony'. Sehingga 'Samsung' dan 'Sony' di dalam Set gabungan hanya muncul 1 kali karena anggota dari Set harus unik. Outputnya adalah sebagai berikut:

```
{'Xiaomi', 'Sony', 'LG', 'Daikin', 'Samsung', 'Apple', 'Panasonic'}
```

Operator Intersection

Contoh penggunaan operator Intersection dapat dilihat pada program berikut:

```
renang = {'siti', 'mail', 'ikhshan', 'upin', 'ipin'}
tenis = {'joko', 'mail', 'ipin', 'upin', 'tejo'}

# suka renang dan tenis
renang_tenis = renang & tenis
print(renang_tenis)
```

Program tersebut akan menghasilkan output hasil operasi intersection dari Set renang dan tenis. Intersection berarti anggota-anggota yang berada di dalam Set renang dan Set tenis sekaligus, yaitu mail, upin dan ipin. Output yang dihasilkan dari program tersebut adalah:

```
{'upin', 'ipin', 'mail'}
```

Operator Difference

Contoh penggunaan operator difference dapat dilihat pada program berikut:

```
# bisa berbahasa english
english = {'desi', 'tono', 'evan', 'miko', 'takashi', 'chaewon'}

# bisa berbahasa korea
korean = {'chaewon', 'yeona', 'erika', 'miko'}

# siapa yang hanya bisa bahasa korea?
only_korean = korean - english
print(only_korean)

# siapa yang hanya bisa bahasa english?
only_english = english - korean
print(only_english)
```

Operator difference akan menghasilkan Set yang anggotanya merupakan selisih dari kedua Set yang dibandingkan. Pada contoh tersebut digunakan untuk mendapatkan anggota yang hanya bisa berbahasa korea dengan cara mencari selisih antara Set korean dan Set english. Sedangkan kebalikannya, jika ingin mengetahui siapa saja yang hanya bisa berbahasa English maka cari selisih antara set English dan Set Korea. Output yang dihasilkan dari program tersebut adalah sebagai berikut:

```
{'erika', 'yeona'} # hanya bisa Korea
{'takashi', 'desi', 'evan', 'tono'} # hanya bisa English
```

Operator Symmetric Difference

Contoh penggunaan operator symmetric difference dapat dilihat pada program berikut ini:

```
english = {'desi', 'tono', 'evan', 'miko', 'takashi', 'chaewon'}
korean = {'chaewon', 'yeona', 'erika', 'miko'}

# hanya bisa bicara satu bahasa saja
one_language = english ^ korean
print(one_language)
```

Operator symmetric difference akan menghasilkan Set baru yang merupakan gabungan dari dua Set tetapi tidak termasuk irisannya. Contoh yang diberikan memanfaatkan operator symmetric difference untuk mendapatkan siapa saja yang hanya dapat berbicara dalam satu bahasa (artinya tidak termasuk irisannya). Secara umum dapat juga dihasilkan dari operasi berikut:

```
one_language = english.union(korean) - english.intersection(korean)
```

Output yang dihasilkan dari program tersebut adalah:

```
{'yeona', 'tono', 'desi', 'erika', 'evan', 'takashi'}
```

12.4 Kegiatan Praktikum

12.4.1 Contoh-contoh Kasus Set

■ Contoh 12.1 Jumlah seluruh elemen yang unik di dalam List

Diberikan sebuah list yang sudah berisi nilai-nilai integer, hitunglah jumlah seluruh elemen list yang unik di dalam list tersebut! Buatlah dalam bentuk fungsi **unique_sum(list)**

■

Pada kasus ini anda diberi satu buah List yang di dalamnya sudah ada beberapa elemen-elemen bertipe integer. Anda diminta untuk menghitung jumlah seluruh elemen yang unik di dalam list tersebut. Sebagai ilustrasi, jika List tersebut sebagai berikut:

```
data = [2, 4, 3, 2, 7, 8, 6, 4, 5, 5]
```

Jumlah seluruh elemen yang unik adalah $2 + 4 + 3 + 7 + 8 + 6 + 5 = 35$. Nilai 2, 4 dan 5 muncul lebih dari sekali, tetapi hanya dihitung satu kali saja (unik). Untuk memecahkan masalah ini, kita akan menggunakan bantuan Set. Solusi untuk kasus ini dapat dilihat pada program berikut ini:

```
def unique_sum(list):
    # ubah dalam bentuk Set
    data_set = set(list)

    # hitung jumlah seluruh anggota data_set
    total = 0
    for data in data_set:
        total = total + data

    # return hasil akhir
    return total

# contoh penggunaan fungsi unique_sum()
contoh1 = [2, 4, 3, 2, 7, 8, 6, 4, 5, 5]
hasil1 = unique_sum(contoh1)
print(hasil1)
```

■ Contoh 12.2 Duplicate chars in string

Buatlah fungsi **cek_duplikat(string)** yang dapat mengecek apakah dalam string terdapat karakter yang muncul lebih dari satu kali (duplikat). Fungsi akan return True jika ada duplikat, dan sebaliknya return False jika tidak ada duplikat karakter dalam string yang diberikan.

■

Anggota dari suatu Set dipastikan selalu unik, sehingga dalam kasus ini kita akan menggunakan Set. Kita akan menggunakan operator **in** untuk mengecek apakah suatu karakter sudah ada di dalam Set atau tidak. Solusi untuk kasus ini dapat dilihat pada program berikut ini:

```
def cek_duplikat(string):
    # buat set kosong
    karakter_set = set()

    # cek semua karakter dalam string satu-persatu
    for karakter in string:
        # apakah karakter ini ada dalam set?
        if karakter in karakter_set:
            # ternyata ada, berarti duplikat
            # hentikan pengecekan, langsung return
            return True
        else:
            # jika belum ada, masukkan dalam set
            karakter_set.add(karakter)

    # setelah looping semua karakter, tidak ada yang sama
    return False

# test case
string1 = 'Alexander the Great' # duplikat
print(cek_duplikat(string1))

string2 = 'UKDW' # semua unik
print(cek_duplikat(string2))
```

■ Contoh 12.3 Kategori aplikasi di Play Store

Buatlah program yang meminta input n kategori aplikasi, lalu program meminta pengguna untuk memasukkan nama-nama aplikasi sebanyak 5 untuk masing-masing kategori. Setelah pengguna selesai memasukkan semua nama aplikasi, program akan menampilkan:

- Daftar nama aplikasi di setiap kategori
- Program yang muncul di semua kategori

■

Untuk membuat program ini, langkah-langkah yang perlu diimplementasikan adalah sebagai berikut:

- Minta jumlah kategori (n).
- Minta nama kategori, kemudian minta nama-nama aplikasi sebanyak 5 untuk kategori tersebut
- Setelah semua kategori selesai di-input, berikutnya tampilkan daftar nama aplikasi di setiap kategori
- Dengan operasi intersection, tampilkan nama aplikasi yang muncul di semua kategori yang ada.

Untuk permasalahan ini, kita akan menggunakan bantuan List, Dictionary dan Set. Solusi untuk masalah tersebut dapat dilihat pada program berikut ini:

```
# input n kategori
n = int(input('Masukkan jumlah kategori: '))
```

```

# siapkan dictionary kosong
data_aplikasi = {}

# input nama kategori dan aplikasi di dalamnya
for i in range(n):
    nama_kategori = input('Masukkan nama kategori:')
    print('Masukkan 5 nama aplikasi di kategori', nama_kategori)

    # siapkan list kosong untuk nama-nama aplikasi
    aplikasi = []
    for j in range(5):
        nama_aplikasi = input('Nama aplikasi: ')
        aplikasi.append(nama_aplikasi)

    # masukkan dalam dictionary
    data_aplikasi[nama_kategori] = aplikasi

# tampilkan dictionary data_aplikasi
print(data_aplikasi)

daftar_aplikasi_list = []
# ambil semua daftar aplikasi dari setiap kategori
for aplikasi in data_aplikasi.values():
    daftar_aplikasi_list.append(set(aplikasi))

print(daftar_aplikasi_list)

# lakukan intersection ke semua set yang ada
hasil = daftar_aplikasi_list[0]
for i in range(1, len(daftar_aplikasi_list)):
    hasil = hasil.intersection(daftar_aplikasi_list[i])

print(hasil)

```

Contoh output yang dihasilkan dari program tersebut adalah sebagai berikut:

```

Masukkan jumlah kategori: 2
Masukkan nama kategori: Finance
Masukkan 5 nama aplikasi di kategori Finance
Nama aplikasi: RTI Saham
Nama aplikasi: Mirae
Nama aplikasi: IPOT
Nama aplikasi: Poems
Nama aplikasi: Calculator
Masukkan nama kategori: Utilities
Masukkan 5 nama aplikasi di kategori Utilities
Nama aplikasi: Photos
Nama aplikasi: Weather

```


Nama aplikasi: Calculator

Nama aplikasi: Camera

Nama aplikasi: Notes

```
{'Finance': ['RTI Saham', 'Mirae', 'IPOT', 'Poems', 'Calculator'],  
'Utilities': ['Photos', 'Weather', 'Calculator', 'Camera', 'Notes']}  
[{'IPOT', 'Poems', 'Calculator', 'RTI Saham', 'Mirae'},  
{'Weather', 'Calculator', 'Camera', 'Photos', 'Notes'}]  
{'Calculator'}
```

Pada contoh tersebut, pengguna memasukkan 2 kategori, yaitu Finance dan Utilities. Daftar nama aplikasi dari setiap kategori tersebut adalah sebagai berikut:

- Finance: RTI Saham, Mirae, IPOT, Poems, Calculator
- Utilities: Photos, Weather, Calculator, Camera, Notes

Dari input user tersebut, kemudian disusun dalam bentuk dictionary sebagai berikut:

```
{  
    'Finance': ['RTI Saham', 'Mirae', 'IPOT', 'Poems', 'Calculator'],  
    'Utilities': ['Photos', 'Weather', 'Calculator', 'Camera', 'Notes']  
}
```

Kemudian dari setiap kategori dilakukan operasi intersection untuk mencari apakah ada nama aplikasi yang muncul di semua kategori. Dari contoh tersebut didapatkan nama aplikasi Calculator yang muncul di semua kategori.

12.5 Latihan Mandiri

Latihan 12.1 Dari contoh kasus kategori kasus di Play Store, tambahkan kemampuan-kemampuan berikut ini:

- Tampilkan nama-nama aplikasi yang hanya muncul di satu kategori saja.
- Untuk input $n > 2$, tampilkan nama-nama aplikasi yang muncul tepat di dua kategori sekaligus

Latihan 12.2 Buatlah sebuah program yang mendemonstrasikan konversi dari:

- List menjadi Set
- Set menjadi List
- Tuple menjadi Set
- Set menjadi Tuple

Tampilkan isi data sebelum dan sesudah konversi.

Latihan 12.3 Buatlah sebuah program yang dapat membaca dua file teks dan menampilkan semua kata-kata yang muncul pada kedua file tersebut. Beberapa hal yang perlu anda perhatikan:

- Nama file adalah input user. Tampilkan pesan error jika file tidak ditemukan/tidak bisa dibaca.
- Semua kata dikonversi dulu menjadi lowercase.
- Sertakan contoh file teks yang anda pakai saat mengumpulkan laporan.



Rekursif dan Regular Expression

13	Fungsi Rekursif	147
13.1	Tujuan Praktikum	
13.2	Alat dan Bahan	
13.3	Materi	
13.3.1	Pengertian Rekursif	
13.3.2	Kelebihan dan Kekurangan	
13.3.3	Bentuk Umum dan Studi Kasus	
13.4	Kegiatan Praktikum	
13.4.1	Problem dan Solusi 1	
13.4.2	Problem dan Solusi 2	
13.4.3	Problem dan Solusi 3	
13.4.4	Problem dan Solusi 4	
13.4.5	Problem dan Solusi 5	
13.5	Latihan Mandiri	
14	Regular Expression	157
14.1	Tujuan Praktikum	
14.2	Alat dan Bahan	
14.3	Materi	
14.3.1	Pengantar Regex	
14.3.2	Meta Character, Escaped Character, Set of Character, dan Fungsi Regex pada Library Python	
14.4	Kegiatan Praktikum	
14.4.1	Penggunaan findall	
14.4.2	Penggunaan search	
14.4.3	Penggunaan split	
14.4.4	Penggunaan sub	
14.5	Latihan Mandiri	
15	Revision History	165
15.1	Daftar Revisi Modul Praktikum	
15.1.1	Semester Genap 2021/2022	
15.1.2	Semester Genap 2019/2020	
	Referensi	167
	Index	169

13. Fungsi Rekursif

13.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan tentang fungsi rekursif.
2. Dapat membaca dan menguraikan cara kerja fungsi rekursif.
3. Dapat menjelaskan beberapa kasus rekursif.
4. Dapat membuat fungsi rekursif.

13.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).

13.3 Materi

13.3.1 Pengertian Rekursif

Fungsi rekursif adalah fungsi yang berisi dirinya sendiri atau fungsi yang mendefinisikan dirinya sendiri. Fungsi ini sering disebut sebagai fungsi yang memanggil dirinya sendiri. Fungsi rekursif merupakan fungsi matematis yang berulang dan memiliki pola yang terstruktur, namun biasanya fungsi ini perlu diperhatikan agar fungsi ini dapat berhenti dan tidak menghabiskan memori. Fungsi

rekursif merupakan fungsi yang harus digunakan secara hati-hati karena fungsi ini dapat bersifat unlimited loop sehingga menyebabkan program hang up.

Fungsi ini akan terus berjalan sampai kondisi berhenti terpenuhi, oleh karena itu dalam sebuah fungsi rekursif perlu terdapat 2 blok penting, yaitu blok yang menjadi titik berhenti dari sebuah proses rekursif dan blok yang memanggil dirinya sendiri. Di dalam rekursif terdapat 2 bagian:

- **Base Case** adalah bagian dimana penentu bahwa fungsi rekursif itu berhenti
- **Rekursif Case** adalah bagian dimana terdapat statement yang akan terus diulang-terus menerus hingga mencapai Base Case

13.3.2 Kelebihan dan Kekurangan

Beberapa keunggulan fungsi rekursif adalah sebagai berikut:

1. Kode program lebih singkat dan elegan.
2. Masalah kompleks dapat di breakdown menjadi sub masalah yang lebih kecil di dalam rekursif

Sedangkan kelemahan fungsi rekursif adalah:

1. Memakan memori yang lebih besar karena setiap kali bagian dirinya dipanggil maka dibutuhkan sejumlah ruang memori tambahan.
2. Mengorbankan efisiensi dan kecepatan.
3. Fungsi rekursif sulit dilakukan debugging dan kadang sulit dimengerti

13.3.3 Bentuk Umum dan Studi Kasus

Bentuk umum fungsi rekursif pada Python:

```
1 def function_name(parameter_list):  
2     ...  
3     function_name(...)  
4     ...
```

Sebenarnya semua fungsi rekursif pasti memiliki solusi iteratifnya. Misalnya pada contoh kasus faktorial berikut: Faktorial adalah menghitung perkalian deret angka $1 \times 2 \times 3 \times \dots \times n$. Algoritma untuk menghitung faktorial adalah:

1. Tanyakan n
2. Siapkan variabel total untuk menampung hasil perkalian faktorial dan set nilai awal dengan 0
3. Loop dari $i = 1$ hingga n untuk mengerjakan:
4. $\text{total} = \text{total} * i$
5. Tampilkan total

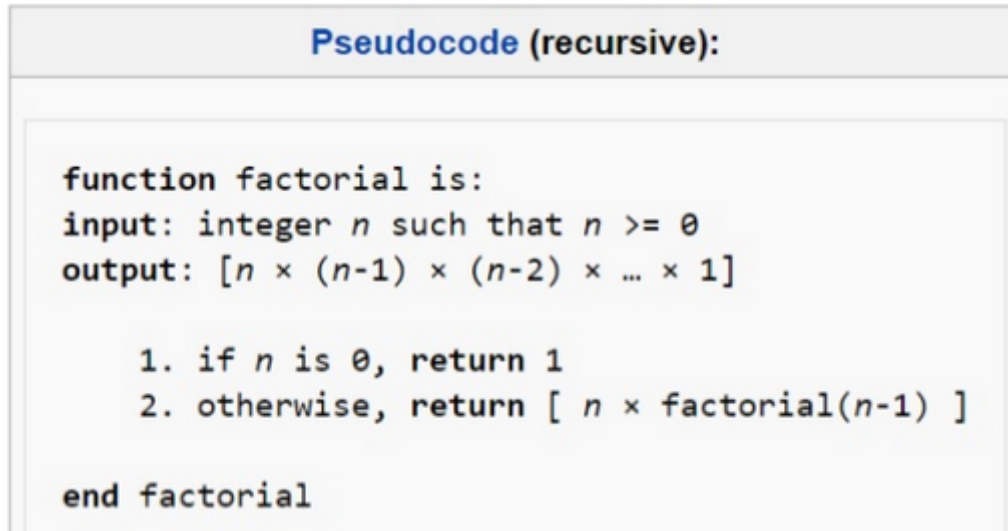
Dengan menggunakan fungsi rekursif maka faktorial dapat dihitung dengan rumus pada gambar 13.1

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{fact}(n - 1) & \text{if } n > 0 \end{cases}$$

Gambar 13.1: Rumus faktorial

Dari rumus 13.1 dapat dibuat pseudocode secara rekursif seperti pada gambar 13.2

Dari pseudocode, maka kode Python yang dapat dibuat adalah:



Gambar 13.2: Pseudocode rekursif faktorial

```
1 def faktorial(n):
2     if n==0 or n==1:
3         return 1
4     else:
5         return faktorial(n-1) * n
6
7 print(faktorial(4))
```

Hasil akhir adalah 24.

Bagaimana proses perhitungan yang terjadi? Berikut adalah gambarannya:

```
1.
2. calc_factorial(4)           # 1st call with 4
3. 4 * calc_factorial(3)       # 2nd call with 3
4. 4 * 3 * calc_factorial(2)   # 3rd call with 2
5. 4 * 3 * 2 * calc_factorial(1) # 4th call with 1
6. 4 * 3 * 2 * 1               # return from 4th call as number=1
7. 4 * 3 * 2                   # return from 3rd call
8. 4 * 6                       # return from 2nd call
9. 24                          # return from 1st call
```

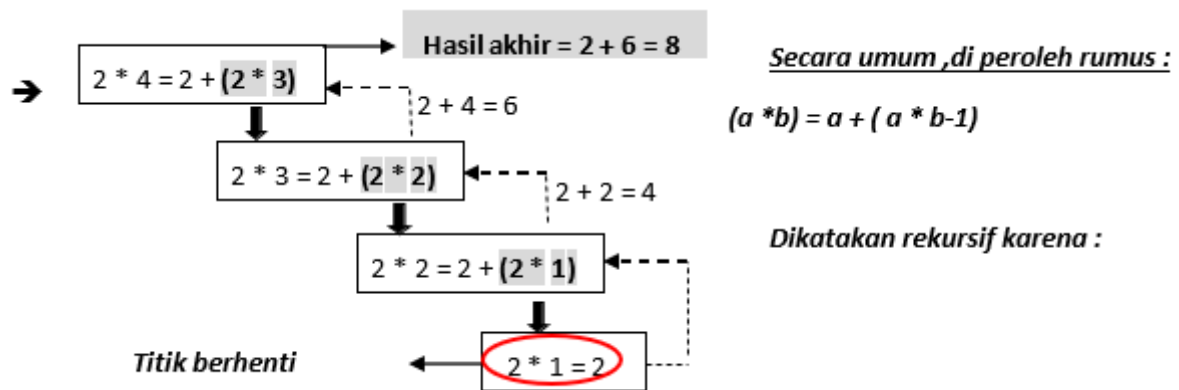
Gambar 13.3: Proses perhitungan faktorial rekursif

13.4 Kegiatan Praktikum

13.4.1 Problem dan Solusi 1

Pada kegiatan praktikum akan dilakukan beberapa percobaan kasus yang dapat diselesaikan dengan menggunakan fungsi rekursif.

Kasus 13.1 Buatlah sebuah program yang dapat melakukan perkalian antara 2 buah bilangan dengan menggunakan fungsi rekursif. Misalkan kita ingin mengalikan angka 2 dengan 4. Dengan metode penjumlahan diperoleh $2 \times 4 = 2 + 2 + 2 + 2 = 8$.



Gambar 13.4: Proses perhitungan perkalian rekursif

Untuk menjawab soal tersebut dapat dilihat logikanya pada gambar:
 Kode program:

```

1  def perkalian(bil1, bil2):
2      if bil2==1:
3          print("%d = " %(bil1), end='')
4          return bil1
5      else:
6          print("%d + " %(bil1), end='')
7          return bil1 + perkalian(bil1, bil2-1)
8
9  print(perkalian(2,4))

```

Hasil: 2 + 2 + 2 + 2 = 8

13.4.2 Problem dan Solusi 2

Kasus 13.2 Buatlah sebuah program yang dapat melakukan pemangkatan antara 2 buah bilangan dengan menggunakan fungsi rekursif. Misalkan kita ingin memangkatkan angka 2 dengan 4. Dengan metode penjumlahan diperoleh $2^{**}4 = 2 * 2 * 2 * 2 = 16$.

Untuk menjawab soal tersebut dapat dilihat logikanya hampir sama dengan 13.4 sebelumnya. Hanya saja operatornya diganti dengan * bukan +.

Kode program:

```

1  def pangkat(bil1, bil2):
2      if bil2==1:
3          print("\%d = " \%(bil1), end='')
4          return bil1
5      else:
6          print("\%d * " \%(bil1), end='')
7          return bil1 * pangkat(bil1, bil2-1)
8
9  print(pangkat(2,4))

```

Hasil: 2 * 2 * 2 * 2 = 16

13.4.3 Problem dan Solusi 3

Kasus 13.3 Tini adalah anak yang pelupa, ia mendapatkan tugas untuk mencari bilangan pada deret Fibonacci dengan urutan tertentu. Dari pada harus selalu menghitung dari awal, bantulah Tono dengan membuat program yang menampilkan bilangan tertentu pada deret Fibonacci sesuai dengan urutan yang diinputkan user. Yang perlu diingat, berikut ini adalah bentuk deret Fibonacci.
1 1 2 3 5 8 13 21 34 ... n

Bilangan fibonacci adalah bilangan yang berasal dari penjumlahan 2 bilangan sebelumnya. Secara iteratif dapat dibuat program sebagai berikut:

```
1 def fibo(n):  
2     f1,f2=1,1  
3     print(f1," ",f2," ",end='')  
4     for i in range(2,n):  
5         fib = f1+f2  
6         f1 = f2  
7         f2 = fib  
8         print(fib," ",end='')  
9  
10    fibo(7)
```

Output adalah: 1, 1, 2, 3, 5, 8, 13

Secara rekursif rumus fibonacci adalah:

$$F(n) = \begin{cases} 1, & n = 1 \text{ dan } 2 \\ F(n-1) + F(n-2), & n > 2 \end{cases}$$

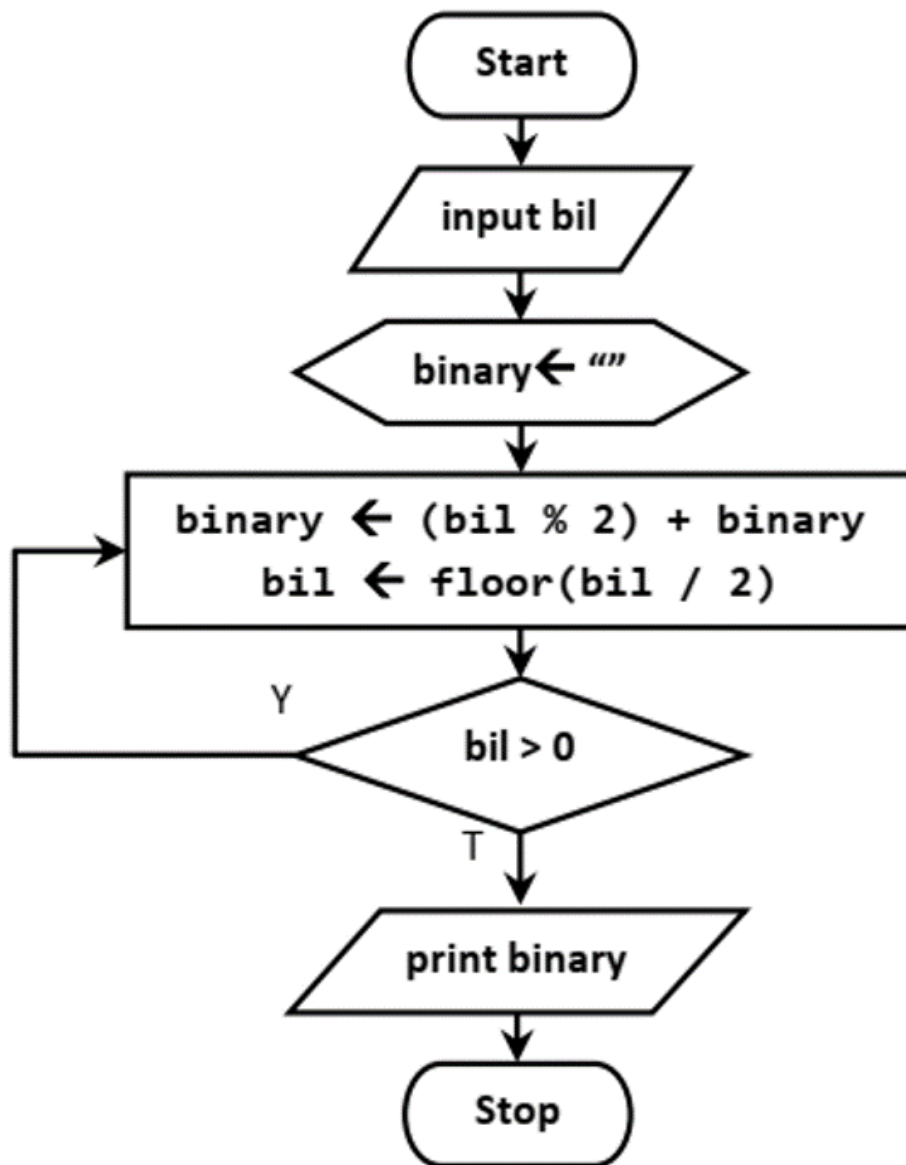
Gambar 13.5: Rumus Fibonacci

```
1 def fibo(n):  
2     if n==1 or n==2:  
3         return 1  
4     else:  
5         return fibo(n-1) + fibo(n-2)  
6  
7     print(fibo(7))
```

Hasil adalah 13

13.4.4 Problem dan Solusi 4

Kasus 13.4 Buatlah program yang dapat mengkonversi suatu bilangan dari basis 10 ke basis lainnya. Input berupa bilangan dalam basis 10 dan basis bilangan (selain basis 10). Program harus dibuat secara rekursif.



Gambar 13.6: Desimal ke Biner

Untuk bisa menyelesaikan masalah ini, anda harus tahu cara mengkonversi basis bilangan dari basis 10 ke basis tertentu. Misalnya diketahui angka adalah 7 dalam basis 10, untuk konversi ke basis 2 dilakukan cara seperti pada gambar 13.6

Sedangkan cara konversi ke basis 8 dilakukan sesuai gambar 13.7

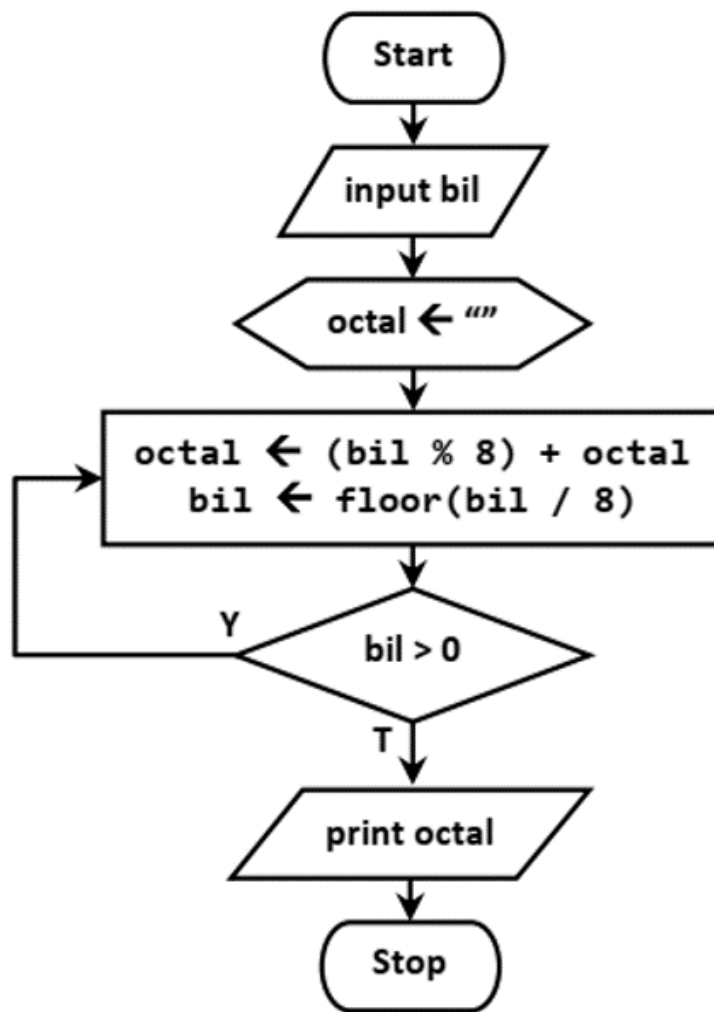
Sedangkan cara konversi ke basis 16 dilakukan sesuai gambar 13.8

Program yang dibuat dapat ditulis secara rekursif sebagai berikut:

```

1 def toBasis(n,base):
2     convertString = "0123456789ABCDEF"
3     if n < base:
4         return convertString[n]
5     else:

```



Gambar 13.7: Desimal ke Oktal

```

6         return toBasis(n//base,base) + convertString[n%base]
7
8     print("Silahkan masukkan bilangan dan basis")
9     angka = int(input("Bilangan : "))
10    basis = int(input("Basis (2/8/16) : "))
11    print(toStr(angka,basis))
  
```

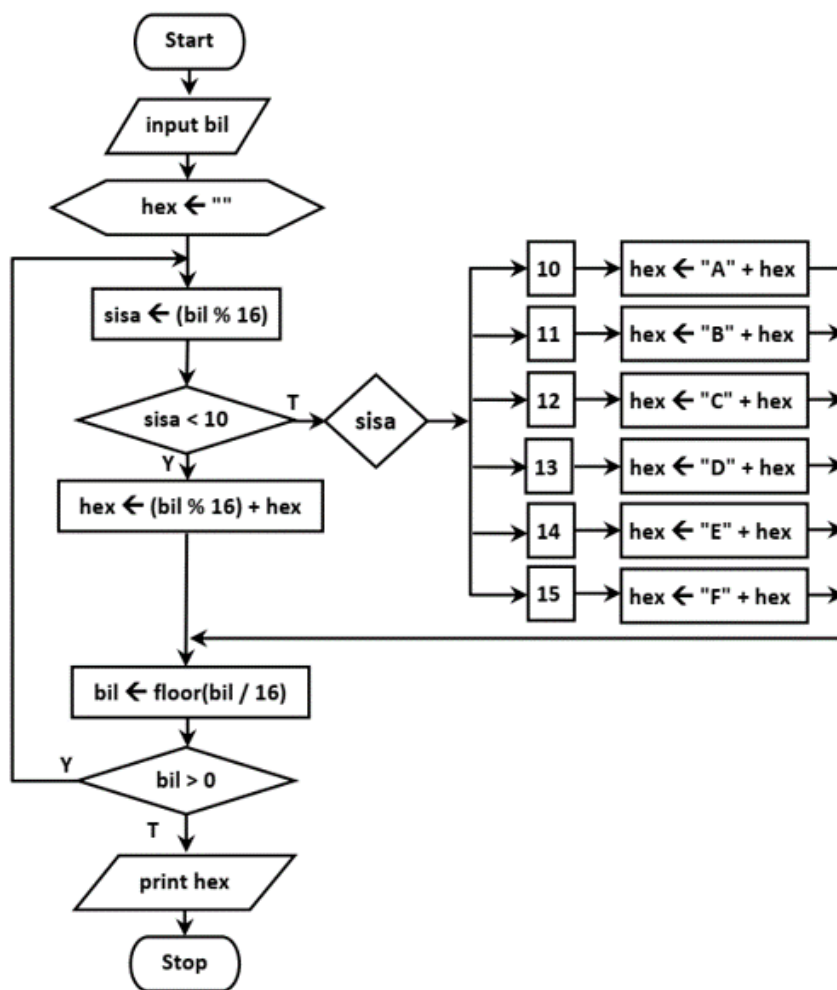
Contoh Output:

```

Bilangan : 9
Basis : 8
11
  
```

Penjelasan:

- Program di atas dimulai dengan memasukkan parameter bilangan dan basisnya dalam integer
- Program akan menyiapkan string yang berisi angka seluruh basis bilangan, dari 0-9, A-F.
- Program akan memeriksa apakah $n < \text{basis}$, jika iya maka kembalikan posisi `string[n]`
- Program akan menjalankan rekursif case dengan cara membagi bilangan / basis dan ditambahkan (dijejarkan) dengan `string[n%basis]`



Gambar 13.8: Desimal ke Hexa

13.4.5 Problem dan Solusi 5

Kasus 13.5 Buatlah program untuk menghitung permutasi secara rekursif!. Permutasi memiliki rumus: $P(n,r) = n! / (n-r)!$

Untuk bisa menjawab soal tersebut kita harus tahu pola menghitung permutasi. Berikut contoh pola menghitung permutasi $P(m,n)$: $P(3,1) = P(3,0) * 3$, jika $n=0$ maka $m P(5,2) = P(5,1) * 4$, jika $n=0$ maka $m P(5,4) = P(5,3) * 2$, jika $n=0$ maka $m P(10,2) = P(10,1) * 9$, jika $n=0$, maka m

Oleh karena itu berikut adalah kode programnya:

```

1 def permutasi (m, n):
2     #m : batas atas dan n : batas bawah, dimana m > n
3     if(n == 0):
4         #jika n = 0, maka hasil adalah m!/(m-1)! = 1
5         return 1
6     else:
7         #jika n > 1 panggil method permutasi secara rekursif dengan parameter n-1 sampai n
8         return (permutasi(m,n-1) * (m-n+1))
9         #kembalikan nilai permuteRec dengan parameter m dan n-1 dikalikan dengan m-n+1
10

```

11 `print(permutasi(10,4))`

13.5 Latihan Mandiri

Latihan 13.1 Vidi adalah adik Tono yang sedang belajar bilangan prima. Vidi mengalami kesulitan untuk menentukan suatu bilangan bilangan prima atau bukan. Untuk membantu adiknya Tono kemudian membuat program untuk pengecekan bilangan prima dengan menggunakan fungsi rekursif. Bantulah Tono untuk menyelesaikan tugas tersebut. ■

Latihan 13.2 Buatlah fungsi rekursif mengetahui suatu kalimat adalah palindrom atau bukan! ■

Latihan 13.3 Buatlah fungsi rekursif untuk menghitung jumlah deret ganjil dari $1 + 3 + 7 + \dots + n!$ ■

Latihan 13.4 Buatlah fungsi rekursif untuk mengetahui jumlah digit dari suatu bilangan. Seperti misalnya tulisan: "234" maka jumlah digitnya adalah $2+3+4 = 9!$ ■

Latihan 13.5 Buatlah fungsi rekursif untuk menghitung kombinasi! ■

14. Regular Expression

14.1 Tujuan Praktikum

Setelah mempelajari Bab ini, mahasiswa diharapkan dapat:

1. Dapat menjelaskan tentang regular expression.
2. Dapat menggunakan simbol dan fungsi regex secara umum.
3. Dapat menggunakan library regex pada Python.
4. Dapat menyelesaikan kasus-kasus regex pada Python.

14.2 Alat dan Bahan

Praktikum ini membutuhkan perangkat komputer yang memiliki spesifikasi minimum sebagai berikut:

1. Terkoneksi ke Internet dan dapat mengunduh package-package Python.
2. Mampu menjalankan sistem operasi Windows 10 atau Ubuntu Linux.

Perangkat lunak yang diperlukan untuk mendukung praktikum ini adalah sebagai berikut:

1. Python 3.7 atau 3.8 yang terinstall menggunakan Anaconda atau Installer Python lainnya.
2. Web Browser (Mozilla Firefox, Microsoft Edge atau Google Chrome).
3. Command Prompt (jika menggunakan Windows).
4. Terminal (jika menggunakan Linux).
5. Editor Python (Visual Studio Code, PyCharm, Spyder atau editor-editor lainnya yang mendukung Python).
6. File mbox-short.txt (di e-class)

14.3 Materi

14.3.1 Pengantar Regex

Pada bab String, kita sudah sedikit mempelajari mengenai teknik-teknik pengaksesan string, manipulasi string, dan berbagai kasus-kasus pengolahan string lainnya, termasuk string yang terdapat pada file. Dari pengalaman tersebut dapat dilihat bahwa kita cukup kesulitan untuk

melakukan pengolahan string dengan teknik biasa / standar. Terdapat teknik pengolahan string yang lebih mudah dan cepat dengan menggunakan bantuan regular expression.

Regular expression adalah ekspresi pola yang berbentuk kumpulan karakter yang digunakan untuk menemukan pola (pattern) yang sama dengan pola regex di dalam string lain yang ingin dicari. Regex membantu kita dalam pencarian string dengan pola tertentu, mengganti string dengan pola tertentu, dan menghapus string dengan pola tertentu. Intinya regex membantu dalam parsing string yang selama ini biasanya hanya menggunakan perintah `split()` dan `find()` saja.

Regex sangat powerful dalam searching dan extracting pola namun memiliki pola yang cukup rumit. Tidak semua bahasa pemrograman mendukung regular expression library. Python merupakan salah satu bahasa yang mendukung library regex dengan cara `import re`. Salah satu fungsi yang paling mudah digunakan dari library `re` adalah `search()`.

Dengan menggunakan file `mbox-short.txt`, kita akan mencoba menampilkan semua string pada file tersebut yang **mengandung** pola "From: ".

```
1 import re
2 handle=open('mbox-short.txt')
3 count = 0
4 for line in handle:
5     line=line.rstrip()
6     if re.search('From:', line):
7         count += 1
8         print(line)
9     print("Count: ",count)
```

Dari kode di atas kita dapat melihat bahwa `re.search` bisa saja diganti dengan menggunakan perintah `find()` pada string biasa. Pola pada contoh di atas belum menggunakan kemampuan regex yang seutuhnya.

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: gsilver@umich.edu
From: gsilver@umich.edu
From: zqian@umich.edu
From: gsilver@umich.edu
From: wagnermr@iupui.edu
From: zqian@umich.edu
From: antranig@caret.cam.ac.uk
From: gopal.ramasammycook@gmail.com
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: stephen.marquard@uct.ac.za
```



```
From: louis@media.berkeley.edu
From: louis@media.berkeley.edu
From: ray@media.berkeley.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
Jumlah: 27
```

Jika diinginkan mencari baris yang **diawali** dengan pola "From", maka kita harus mengubah parameter fungsi search pada re.search menjadi **re.search("^From")**.

```
1 import re
2 handle=open('mbox-short.txt')
3 count = 0
4 for line in handle:
5     line=line.rstrip()
6     if re.search('^From:', line):
7         count += 1
8         print(line)
9 print("Count: ",count)
```

Kode di atas dapat digantikan juga dengan fungsi string.startswith("From :").

14.3.2 Meta Character, Escaped Character, Set of Character, dan Fungsi Regex pada Library Python

Sebelum menggunakan fungsi regex perlu diketahui terlebih dahulu meta character / special character dan kegunaannya pada pola regex seperti pada tabel 14.1

Pada Python terdapat beberapa special character (escaped characters) seperti pada tabel 14.2

Pada Python terdapat beberapa penggunaan himpunan character dengan menggunakan simbol [], pada tabel 14.3

Pada Python terdapat 4 buah fungsi yang bisa dipakai untuk menggunakan Regex seperti pada tabel 14.4

14.4 Kegiatan Praktikum

14.4.1 Penggunaan findall

Kita akan mencoba penggunaan fungsi **findall** untuk mencari semua pola sebagai berikut:

```
1 import re
2
3 txt = "Sang mata-mata sedang memata-matai kasus kaca mata di toko Matahari"
4 x = re.findall("mata", txt)
5 y = re.findall("saya", txt)
6 for i in x:
7     print(i)
8
9 if (y):
10     print("Ada yang cocok!")
```

Tabel 14.1: Special Character pada Python

Karakter	Kegunaan	Contoh	Arti Contoh
[]	Kumpulan karakter	"[a-zA-Z]"	1 karakter antara a-z kecil atau A-Z besar
\{ }	Karakter dengan arti khusus dan escaped character	\{ }d	Angka / digit
.	Karakter apapun kecuali newline	say.n.	Tidak bisa diganti dengan karakter apapun, misal "sayang" akan valid
^	Diawali dengan	^From	Diawali dengan From
\$	Dakhiri dengan	this\$	Diakhiri dengan kata this
*	0 s/d tak terhingga karakter	\{ }d*	ada digit minimal 0 maksimal tak terhingga
?	ada atau tidak (opsional)	\{ }d?	Boleh ada atau tidak ada digit sebanyak
+	1 s/d tak terhingga karakter	\{ }d+	Minimal 1 s/d tak terhingga karakter
{ }	Tepat sebanyak yang ada para { }	\{ }d{2}	Ada tepat 2 digit
()	Pengelompokan karakter / pola	(sayalkamu)	saya atau kamu sebagai satu kesatuan
	atau	\{ }d \{ }s	1 digit atau 1 spasi

Tabel 14.2: Escaped Character pada Regex

Special Characters	Kegunaan	Contoh
\b	Digunakan untuk mengetahui apakah suatu pola berada di awal kata atau akhir kata	"R\b in" "Rain\b"
\d	Digunakan untuk mengetahui apakah karakter adalah sebuah digit (0 s/d 9)	\d
\D	Digunakan untuk mengetahui apakah karakter yang bukan digit	\D
\s	Digunakan untuk mengetahui apakah karakter adalah whitespace (spasi, tab, enter)	\s
\S	Digunakan untuk mengetahui apakah karakter adalah BUKAN whitespace (spasi, tab, enter)	\S
\w	Digunakan untuk mengetahui apakah karakter adalah word (a-z, A-Z, 0-9, dan _)	\w
\W	Digunakan untuk mengetahui apakah karakter adalah BUKAN word (a-z, A-Z, 0-9, dan _)	\W
\A	Digunakan untuk mengetahui apakah karakter adalah berada di bagian depan dari kalimat	"\AThe"
\Z	Digunakan untuk mengetahui apakah karakter adalah berada di bagian akhir dari kalimat	"End\Z"

```

11 else:
12     print("Tidak ada yang cocok!")

```

Tabel 14.3: Himpunan Karakter pada Regex

[abc]	Mencari pola 1 huruf a, atau b, atau c
[a-c]	Mencari pola 1 huruf a s/d c
[^bmx]	Mencari pola 1 huruf yang bukan b,m, atau x
[012]	Mencari pola 1 huruf 0, atau 1, atau 2
[0-3]	Mencari pola 1 huruf 0 s/d 3
[0-2][1-3]	Mencari pola 2 huruf: 01, 02, 03, 11, 12, 13, 21, 22, 23
[a-zA-Z]	Mencari pola 1 huruf a-Z

Tabel 14.4: Fungsi Regex pada Python

Nama Fungsi	Kegunaan
findall	mengembalikan semua string yang sesuai pola (matches)
search	mengembalikan string yang sesuai pola (match)
split	memecah string sesuai pola
sub	mengganti string sesuai dengan pola yang cocok

Hasil:

```
mata #pada mata-mata
mata #pada mata-mata
mata #pada memata-matai
mata #pada memata-matai
mata #kaca mata
Tidak ada yang cocok #karena tidak ada 'saya'
```



Perhatikan bagian Matahari tidak muncul karena Matahari menggunakan huruf besar

Contoh lain fungsi **findall**:

```
1 import re
2 handle=open('mbox-short.txt')
3 for line in handle:
4     line=line.rstrip()
5     x=re.findall('\S@\S+', line)
6     if len(x)>0:
7         print(x)
```

Hasil:

```
['stephen.marquard@uct.ac.za']
['<postmaster@collab.sakaiproject.org>']
['<200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
```

```
['source@collab.sakaiproject.org;']  
['stephen.marquard@uct.ac.za']  
['source@collab.sakaiproject.org']  
....dst
```



Ada beberapa format email yang tidak sesuai format, seperti mengandung karakter <, sehingga kita perlu mengganti format regex nya menjadi: [a-zA-Z0-9]\${}*@[a-zA-Z. Silahkan ubah dibagian baris ke-5.

14.4.2 Penggunaan search

Kita akan mencoba penggunaan fungsi **search** untuk mencari pola sebagai berikut:

```
1 import re  
2  
3 txt = "Sang mata-mata sedang memata-matai kasus kaca mata di toko Matahari"  
4 x = re.search("\s", txt)  
5 y = re.search("saya", txt)  
6  
7 print("Spasi ditemukan di:", x.start())  
8 print(y)
```

Hasil:

```
4  
None
```

Contoh lain fungsi **search**: Pada mbox kita ingin menemukan kata-kata:

```
X-DSPAM-Confidence: 0.847  
5X-DSPAM-Probability: 0.0000  
X-DSPAM-Confidence: 0.6178  
X-DSPAM-Probability: 0.0000
```

Untuk melakukannya dapat digunakan regex: $\hat{X}-.*: [0-9.]+$

```
1 import re  
2 handle=open('mbox-short.txt')  
3 for line in handle:  
4     line=line.rstrip()  
5     if(re.search('^X-.*: [0-9.]+', line)):  
6         print(line)
```

Hasil:

```
X-DSPAM-Confidence: 0.8475  
X-DSPAM-Probability: 0.0000  
X-DSPAM-Confidence: 0.6178  
X-DSPAM-Probability: 0.0000  
X-DSPAM-Confidence: 0.6961  
X-DSPAM-Probability: 0.0000  
X-DSPAM-Confidence: 0.7565  
X-DSPAM-Probability: 0.0000  
dst...
```

14.4.3 Penggunaan split

Kita akan mencoba penggunaan fungsi **split** untuk memecah string sebagai berikut:

```
1 import re
2
3 txt = "The rain in Spain"
4 x = re.split("\s", txt)
5 print(x)
6 y = re.split("\s", txt, 1) #split 1 kata pertama
7 print(x)
```

Hasil:

```
['The', 'rain', 'in', 'Spain']
['The', 'rain in Spain']
```

14.4.4 Penggunaan sub

Kita akan mencoba penggunaan fungsi **sub** untuk replace pola sebagai berikut:

```
1 import re
2
3 txt = "Sang mata-mata sedang memata-matai kasus kaca mata di toko Matahari"
4 x = re.sub("\s", "-", txt) #mengganti spasi dengan -
5 print(x)
6 y = re.sub("\s", "*", txt, 2) #mengganti spasi dengan * 2 saja
7 print(y)
```

Hasil:

```
Sang-mata-mata-sedang-memata-matai-kasus-kaca-mata-di toko-Matahari
Sang*mata-mata*sedang memata-matai kasus kaca mata di toko Matahari
```

14.5 Latihan Mandiri

Latihan 14.1 Anda diminta untuk mencari seluruh teks yang berupa tanggal dengan format YYYY-MM-DD dan kemudian seluruh tanggal tersebut diambil dan ditampilkan kembali dalam format DD-MM-YYYY ditambah dengan perhitungan selisih dengan tanggal sekarang dalam hari.

Contoh:

Pada tanggal 1945-08-17 Indonesia merdeka. Indonesia memiliki beberapa pahlawan nasional, seperti Pangeran Diponegoro (TL: 1785-11-11), Pattimura (TL: 1783-06-08) dan Ki Hajar Dewantara (1889-05-02).

Hasil:

```
1945-08-17 00:00:00 selisih 27209 hari
1785-11-11 00:00:00 selisih 85561 hari
1783-06-08 00:00:00 selisih 86448 hari
1889-05-02 00:00:00 selisih 47769 hari
```

Latihan 14.2 Anda diminta untuk mencari seluruh teks yang berupa email dan kemudian ambil semua username dari email tersebut untuk digenerate password random 8 karakter yang terdiri dari angka dan huruf.

Contoh:

Berikut adalah daftar email dan nama pengguna dari mailing list:

anton@mail.com dimiliki oleh antonius

budi@gmail.co.id dimiliki oleh budi anwari

slamet@getnada.com dimiliki oleh slamet slumut

matahari@tokopedia.com dimiliki oleh toko matahari

Hasil:

anton@mail.com username: anton , password: 8u78A2UD

budi@gmail.co.id username: budi , password: bdP066Ld

slamet@getnada.com username: slamet , password: Ab1FiHXb

matahari@tokopedia.com username: matahari , password: 5KYyaP6

15. Revision History

15.1 Daftar Revisi Modul Praktikum

15.1.1 Semester Genap 2021/2022

Pada semester ini dilakukan beberapa revisi besar dan pembaruan materi dari bab-bab berikut ini:

- Bab 1: perbaikan penulisan, pembaruan gambar, penambahan materi dan perubahan soal Latihan mandiri. Ditulis oleh Yuan Lukito.
- Bab 2: perubahan pada soal 01 Latihan Mandiri, perbaikan penulisan soal 02 dan perbaikan penulisan formula matematika. Ditulis oleh Yuan Lukito.
- Bab 5: perubahan susunan materi, penambahan pembahasan mengenai break dan continue, dan penambahan soal Latihan Mandiri.

Penulis: Yuan Lukito (Penulis pertama), Laurentius Kuncoro P. S. dan Matahari Bhakti Nendya. (Penulis kedua dan ketiga).

15.1.2 Semester Genap 2019/2020

Merupakan semester pertama modul praktikum ini digunakan. Berisi 14 materi yang disusun oleh Yuan Lukito, Antonius Rachmat, Laurentius Kuncoro P. S. dan Matahari Bhakti Nendya.

Penulis: Yuan Lukito (Penulis pertama), Laurentius Kuncoro P. S. dan Matahari Bhakti Nendya. (Penulis kedua, ketiga dan ke-empat).

Bibliografi

- [1] Severance, C.R. (2016) *Python for Everybody*. CreateSpace Independent Publishing Platform
- [2] Rachmat, A. (2011) *Algoritma dan Pemrograman dengan Bahasa C*. Penerbit Andi, Yogyakarta

Indeks

A

Anaconda	3
Argument	51

B

Boolean	36
Break	73
Bug	7

C

C	2
Continue	74

D

Dictionary	107
------------------	-----

E

Ekspresi	23
----------------	----

F

File	91
Fungsi	47

I

Interpreter Python	6
--------------------------	---

J

Java	2
------------	---

K

keywords	21
komentar	26

L

Lambda	52
List	101
Logo Python	2

O

Operator dan Operand	22
----------------------------	----

P

Parsing	87
Percabangan	37
Percabangan Kompleks	67
Perulangan Bertingkat	75
Program	91
pustaka	2
Python	2

R

Regex	157
-------------	-----

Rekursif	147
Return	49

S

Set	133
Statement	22
String	83

T

Tuple	120
type	19

V

value	19
variabel	20
volatile	92